6 Les instructions

PLAN

OBJECTIFS

- Maîtriser les traitements sur les données et en particulier les instructions répétitives qui exploitent la puissance de l'ordinateur.

6.1 Introduction

Ce chapitre décrit les traitements appliqués aux données, les instructions.

Nous présentons les instructions simples, parmi lesquelles on trouve l'instruction d'affectation et le problème de priorité des opérateurs dans l'évaluation d'une expression, l'instruction de branchement inconditionnel ou goto et l'appel de procédure.

Nous abordons ensuite les instructions composées, éléments importants des mécanismes de programmation. On y retrouve les instructions conditionnelles ou tests qui effectuent un traitement selon un test logique ou un choix multiple, et les instructions répétitives ou boucles qui donnent accès à la puissance de calcul de l'ordinateur.

Enfin nous terminons par les instructions de contrôle.

6.2 Les instructions simples

Présentation

Les instructions simples regroupent l'affectation, l'appel de procédure, l'instruction goto. Contrairement aux instructions composées, elles s'écrivent sur une seule ligne et ne contiennent aucune autre instruction.

L'instruction d'affectation =

Forme générale

L'affectation est représentée par le signe égal « = ». Sa forme générale est :

```
$variable = expression ;
```

où « expression » peut être un calcul arithmétique, logique, ou un appel de fonction, comme le montrent les syntaxes suivantes :

```
$i = $i+1                      ; // Calcul arithmétique
$Est_Majeur = ($age >= 18)     ; // Calcul logique
$Surface = 4*PI*pow($Rayon,2) ; // Appel d'une fonction
```

Si la syntaxe de cette instruction est simple, son exécution peut engendrer des calculs erronés selon l'ordre d'évaluation et le type des données. Il est important de connaître son fonctionnement en deux étapes :

1. *Evaluation* de l'expression de droite.
   L'évaluation est effectuée avant l'affectation. Dans l'instruction `$i=$i+1`, on calcule d'abord `$i+1`, puis on affecte le résultat à `$i`. Les deux opérations sont dissociées. Le résultat de l'évaluation dépend des *règles de priorité entre les opérateurs*.
2. *Affectation* du résultat à la variable de gauche.
   La variable contient le résultat du calcul et *prend le type du résultat*.

Règles de priorité

Les règles de priorité entre opérateurs impactent l'ordre du calcul, donc le résultat obtenu. Par exemple, **2 + 8 * 3** donne le résultat **26** et non **30**, car la multiplication « * » a une priorité supérieure à l'addition « + ».

Lorsque les opérateurs ont une priorité égale, leur association est évaluée, soit par la gauche, soit par la droite, selon l'opérateur.

Par exemple, « – » est une association par la gauche, ainsi **2 – 4 – 6** est évalué en **(2 – 4) – 6**. Alors que « = » est une association par la droite, ainsi, **$a = $b = $c** est groupé en : **$a = ($b = $c)**.

Les opérateurs de priorité égale non associatifs (l'ordre est important) ne peuvent pas être utilisés entre eux.

Par exemple, **1 < 2 > 1 est illégal** en PHP, alors que **1 <= 1 == 1 est autorisé**, car l'opérateur == possède une priorité inférieure à l'opérateur <=.

Le tableau 6.1 présente la priorité des opérateurs, du plus prioritaire (en haut) au moins prioritaire (en bas). Pour les opérateurs de même priorité, l'associativité (regroupement) par la gauche ou par la droite est indiquée.

Tableau 6.1 – Priorité entre opérateurs

Associativité	Opérateur	Catégorie
non associatif	`clone new`	clone et new
gauche	`[`	tableaux
droite	`**`	opérateurs arithmétiques
droite	`++ -- ~ (int) (float) (string)(array) (object) (bool) @`	types, incrément, décrément
non associatif	`instanceof`	types
droite	`!`	opérateurs logiques
gauche	`* / %`	opérateurs arithmétiques
gauche	`+ - .`	opérateurs arithmétiques et chaînes de caractères
gauche	`<< >>`	opérateurs binaires
non associatif	`< <= > >=`	opérateurs de comparaison
non associatif	`== != === !== <>`	opérateurs de comparaison
gauche	`&`	opérateurs binaires et référence
gauche	`^`	opérateurs binaires
gauche	`|`	opérateurs binaires
gauche	`&&`	opérateurs logiques
gauche	`||`	opérateurs logiques
gauche	`? :`	opérateurs ternaires
droite	`= += -= *= **= /= .= %= &= |= ^= <<= >>= =>`	opérateurs d'affectation
gauche	`and`	opérateurs logiques
gauche	`xor`	opérateurs logiques
gauche	`or`	opérateurs logiques
gauche	`,`	plusieurs utilisations possibles

Cependant, il est parfois difficile de comprendre l'ordre d'évaluation d'une expression complexe, et cela peut être dangereux de laisser l'interpréteur choisir cet ordre, au risque d'obtenir un résultat différent de celui qui est attendu.

Il est préférable de mettre des parenthèses pour préciser l'ordre d'évaluation.

Ainsi, à la place de l'expression :

```
$t = $x + $y / $a * $z - $y / $b ;
```

il est préférable d'écrire avec les parenthèses l'ordre souhaité :

```
$t = ($x + $y) / (($a * $z) - ($y / $b)) ;
```

Règles de « transtypage »

Lors d'un calcul, le *résultat* est implicitement typé, ce qui peut changer le type de la variable qui le reçoit. Avec une expression arithmétique contenant des entiers et des réels, le résultat sera un réel, soit le plus grand type commun. Lorsque l'expression mélange chaînes de caractères et numériques, PHP exprime le tout en numérique. Le tableau 6.2 détaille ces règles de transtypage *implicites*.

Tableau 6.2 – Règles de transtypage implicites

Type opérande 1	Type opérande 2	Type du Résultat
Null ou chaîne de caractères	Chaînes de caractères	Convertit NULL en " ", puis comparaison numérique ou ASCII
Chaînes de caractères, ressources ou nombre	Chaînes de caractères, ressources ou nombre	Transforme en nombres avant comparaison
Booléen ou null	N'importe quel type	Convertit en booléen (par définition FALSE < TRUE)
Objet	Objet	Selon les méthodes de comparaison définies par les classes
Objet	N'importe quel type (sauf tableau)	L'objet est toujours plus grand
Tableau	Tableau	Le tableau avec le moins de membres est le plus petit
Tableau	N'importe quel type (sauf objet)	Le tableau est toujours plus grand

L'instruction *goto*

L'instruction `goto` est un *branchement inconditionnel*. Elle utilise une *étiquette* de ligne sur laquelle aboutit le branchement. La syntaxe générale est :

```
goto  label ;
```

où label est l'étiquette d'une ligne.

```
label: $y=$x+3 ;
```

Le programme `goto_shell.php` en présente un exemple.

Listing 6.1 – Programme goto_shell.php

```
<?php
saisie : echo "Entrez deux entiers : ";
fscanf(STDIN,"%d %d",$i,$j);
if ($j == 0)
{echo "Erreur : la 2ème valeur ne doit pas être 0".PHP_EOL;
 goto saisie ;
}
else echo "résultat de $i/$j = ",$i/$j,PHP_EOL;
?>
```

Voici son exécution :

Listing 6.2 – Exécution de goto_shell.php

```
$ php goto_shell.php
Entrez deux entiers : 12 0
Erreur : la 2ème valeur ne doit pas être 0
Entrez deux entiers : 12 3
résultat de 12/3 = 4
```

L'instruction `goto` est incompatible avec une programmation structurée. Elle redirige l'exécution du programme vers les lignes suivantes ou précédentes, et annule la logique d'exécution, ligne par ligne, du haut vers le bas. Avec plusieurs `goto`, il est difficile de suivre le déroulement logique du programme.

Remarque

Les instructions goto sont à prohiber. Elles déstructurent le programme. Elles peuvent toujours être remplacées par des tests bien écrits ou des boucles.

L'appel de procédure ou de fonction

Lorsqu'une instruction de la forme `proc($x,$y,…)` est rencontrée, PHP l'interprète comme une procédure. De même, la syntaxe `$res=fonction($x,$y,…)` est analysée comme l'appel d'une fonction. Ces deux instructions simples ne présentent aucune difficulté syntaxique.

Cette section présente le *formatage de l'affichage* et de *la saisie* en mode *shell via* `printf()` et `fscanf()`. La présentation des données dans un environnement web utilise d'autres fonctions telles que `number_format()` qui sont étudiées dans le chapitre sur les chaînes de caractères.

Affichage formaté en shell *printf()*

- *Forme générale*

L'instruction `printf()` est héritée du langage C. Sa forme générale est :

```
printf(format,$variable1,$variable2,...) ;
```

où `$variable1,$variable2,…` est une liste de variables et `format` une chaîne de caractères précisant le format d'affichage pouvant contenir des caractères spéciaux (`\n` = saut de ligne), et des *codages de type* spécifiant le type à utiliser pour les variables à afficher. La syntaxe suivante affiche trois entiers au format décimal (`%d`). La ligne est terminée par un saut de ligne (`\n`).

```
$somme=$i+$j;
printf("La somme de %d et de %d = $somme\n",$i,$j,$somme);
```

Avec les valeurs respectives 3 et 4 pour `$i` et `$j`, la syntaxe précédente produit :

```
La somme de 3 et de 4 = 7
```

- *Codage de type*

Le tableau 6.3 résume les différents codages de type.

Tableau 6.3 – printf() : codage des types

Format	Type	Syntaxe	Affichage
d	Entier en décimal	`$i=28 ;` `printf("i=%d",$i);`	`i=28`
o	Entier en octal	`$i=28 ;` `printf("i=%o",$i);`	`i=34`
x,X	Entier en hexadécimal	`$i=28 ;` `printf("i=%X",$i);`	`i=1C`
u	Entier non signé	`$i=-28 ;` `printf("i=%u",$i);`	`i=` `18446744073709551588`
b	Entier en binaire	`$i=28 ;` `printf("i=%b",$i);`	`i=11100`
c	Entier affiché en caractère	`$i=65 ;` `printf("i=%c",$i) ;`	`i=A`
f, F	Réel : notation 23.123456	`$x=2.3e-4 ;` `printf("x=%f",$x);`	`x=0.000230`
s	Chaîne de caractères	`ch="bonjour";` `printf("ch=%s",$ch);`	`ch=bonjour`
e, E	Réel : notation 2.312345e+1 ou 2.312345E+1	`$x=2.3e-4 ;` `printf("x=%E",$x);`	`x=2.300000E-04`

g, G	Correspond soit à %e, %E soit à %f selon la valeur.
	Les zéros de fin ne sont pas affichés
		`$x=2.3e-4 ;`
		`printf("x=%G\n",$x);`
		`$y=2.3e-300 ;`
		`printf("x=%G\n",$y);`
			`x=0.00023`
			`x=2.3E-300`

%	Affiche le caractère %	`printf("caract %%");`	`caract %`

- *Largeur et précision*

Le **codage de type** peut indiquer un affichage plus complet, avec la largeur d'affichage et le nombre de décimales après la virgule. La figure 6.1 présente cette syntaxe et l'affichage obtenu avec la valeur `25.4` pour la variable `$x`.

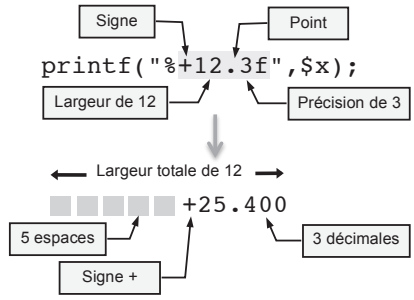


Figure 6.1 – printf() avec format largeur et précision.

Les valeurs des différents éléments du format sont :

- **Signe**
  - ◇ – Précise un cadrage à gauche de l'affichage.
  - ◇ + Le nombre affiché sera toujours signé.
  - ◇ L'espace Si le 1er caractère n'est pas un signe, l'espace est affiché en préfixe.
  - ◇ 0 (zéro) Pour l'affichage numérique. Affiche des 0 de remplissage.
  - ◇ '#, '?, … Caractère de remplissage personnalisé précédé par '.
- **Largeur**
  - ◇ Un nombre Largeur minimale d'affichage. Si cette largeur est trop petite, l'affichage prend la largeur supérieure nécessaire.
- **.** Le point décimal.
- **Précision**
  - ◇ Un nombre La précision maximale utilisée. Si la précision est plus grande, le reste est tronqué à l'affichage.

- *Exemples d'affichage formaté*

Voici quelques exemples de présentation selon le type de la variable.

```
$i = 24; // Entier
printf("|%d|\n",    $i);     //   |24|
printf("|%5d|\n",   $i);     //   |   24|
printf("|%05d|\n",  $i);     //   |00024|
printf("|%-5d|\n",  $i);     //   |24   |
printf("|%'#5d|\n", $i);     //   |###24|

$y = 25.4234 ; // Réel
printf("|%f|\n",       $y);      //     |25.423400|
printf("|%10f|\n",     $y);      //     | 25.423400|
printf("|%10.3f|\n",   $y);      //     |    25.423|
printf("|%+10.3f|\n",  $y);      //     |   +25.423|
printf("|%010.3f|\n",  $y);      //     |000025.423|
printf("|%-10.3f|\n",  $y);      //     |25.423    |
printf("|%'#10.3f|\n", $y);      //     |####25.423|

$s = 'bateaux'; // Chaîne de caractères
$t = 'nombreux bateaux';
printf("|%s|\n",      $s);     //    |bateaux|
printf("|%10s|\n",    $s);     //    |   bateaux|
printf("|%-10s|\n",   $s);     //    |bateaux   |
printf("|%010s|\n",   $s);     //    |000bateaux|
printf("|%'#10s|\n",  $s);     //    |###bateaux|
printf("|%12s|\n", $t)   ;     //    |nombreux bateaux|
printf("|%12.12s|\n", $t);     //    |nombreux bat|
```

Saisie au clavier en shell fscanf()

• *Forme générale*

L'instruction `fscanf()` est héritée du langage C. Sa forme générale est :

```
fscanf(STDIN,format,$variable1,$variable2,...)
```

où `STDIN` est le fichier d'entrée standard correspondant au clavier, `$variable1,$variable2,...` est une liste de variables et `format` est une chaîne de caractères précisant le *format de saisie*.

Le *format de saisie* est une chaîne de caractères contenant : des espaces ou tabulations (ignorés), des caractères ordinaires (sauf %) qui imposent de les saisir, des *codages de type* pour les variables. La syntaxe suivante saisit une date au format « jj/mm/aaaa ». Le jour, le mois et l'année sont affectés à `$jour`, `$mois`, `$annee`.

```
echo "Entrez une date (04/01/2015) : " ;
fscanf(STDIN,"%d/%d/%d",$jour,$mois,$annee) ;
```

• *Codage de type*

Les codages de types sont ceux du `printf()` présentés dans le tableau 6.3.

• *Limitation de la taille de la saisie*

Le *codage de type* peut limiter la largeur de saisie. Avec le format de la figure 6.2, si l'on saisit 123456, la variable `$i` vaut 1234, les chiffres non lus sont perdus.

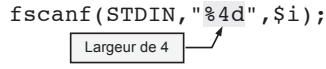


Figure 6.2 – fscanf() avec format largeur.

• *Code de retour*

`fscanf()` utilisée comme une *fonction* retourne un tableau de valeurs, s'il n'y a que deux paramètres, ou le nombre de valeurs lues dans les autres cas. Le programme `saisie_date_fsanf_shell.php` présente ces deux syntaxes :

Listing 6.3 – Programme saisie_date_fsanf_shell.php

```
<?php
 echo "Entrez une date (04/01/2016) : " ;
 $retour=fscanf(STDIN,"%d/%d/%d",$jour,$mois,$annee)  ;
 echo "jour  = $jour\n"     ;
 echo "mois  = $mois\n"     ;
 echo "année = $annee\n"    ;
 var_dump($retour);
 echo "Entrez une nouvelle date (04/01/2016) : " ;
$retour=fscanf(STDIN,"%d/%d/%d");
```

```
 echo "jour  = $retour[0]\n"     ;
 echo "mois  = $retour[1]\n"     ;
 echo "année = $retour[2]\n"     ;
 var_dump($retour);
?>
```

Voici son exécution :

Listing 6.4 – Exécution de saisie_date_fsanf_shell.php

```
$ php saisie_date_fscanf.php
Entrez une date (04/01/2016) : 22/09/2015
jour  = 22
mois  = 9
année = 2015
int(3)
Entrez une nouvelle date (04/01/2016) : 10/02/2016
jour  = 10
mois  = 2
année = 2016
array(3) {
  [0]=>
  int(10)
  [1]=>
  int(2)
  [2]=>
  int(2016)
}
```

6.3 Les instructions composées

Présentation

Les instructions composées contiennent d'autres instructions qui peuvent être simples ou composées. Il y a alors imbrication d'instructions. Elles sont réparties en : instructions conditionnelles ou tests et instructions répétitives ou boucles.

La séquence d'instructions *{}*

Présentation

La séquence d'instructions est un regroupement d'instructions défini par les accolades `{}`. Elle autorise plusieurs instructions dans les tests ou dans les boucles qui n'en permettent qu'une seule. Les instructions regroupées sont vues par le niveau supérieur, comme une unique instruction.

Forme générale

La syntaxe d'une séquence est la suivante :

```
{
  instruction1 ;
  instruction2 ;
  ...          ;
}
```

Les instructions `instruction1`, `instruction2`, `...` sont regroupées en une seule instruction. Voici un exemple d'utilisation :

```
if ($age < 16)
{
 echo "vous êtes mineur".PHP_EOL           ;
 echo "vous êtes encore à l'école".PHP_EOL ;
}
```

Les accolades ne sont jamais suivies de point-virgule. Les instructions composées n'autorisant qu'une seule instruction interne impliquent un nombre élevé de séquences. L'indentation du programme (présentation en décalant de quelques espaces les instructions internes) devient indispensable pour visualiser les niveaux d'imbrication.

Forme alternative

Le regroupement d'instructions possède une autre syntaxe dans laquelle :

- l'accolade ouvrante « `{` » est remplacée par « `:` » ;
- l'accolade fermante « `}` » est remplacée par l'un des mots-clefs suivants :
  - `endif ;` pour l'instruction composée `if`
  - `endswitch ;` pour l'instruction composée `switch`
  - `endwhile ;` pour l'instruction composée `while`
  - `endfor ;` pour l'instruction composée `for`
  - `endforeach ;` pour l'instruction composée `foreach`

L'instruction `if` précédente se réécrit :

```
if ($age < 16) :
 echo "vous êtes mineur".PHP_EOL           ;
 echo "vous êtes encore à l'école".PHP_EOL ;
endif;
```

Les instructions conditionnelles ou tests

Présentation

Les instructions conditionnelles exécutent un bloc d'instructions ou un autre selon le résultat d'un test. Elles sont aussi appelées *branchements conditionnels*.

Deux instructions existent : l'instruction `if…else` et ses variantes et l'instruction `switch`.

L'instruction if…else et ses variantes

L'instruction `if…else` utilise un *test booléen*. Elle est employée quand deux choix sont possibles. Elle présente deux versions : l'instruction `if…else`, l'instruction `if`.

- *L'instruction if … else*

Forme générale

La forme générale de l'instruction est :

```
if (test_booleen)
  instruction1  ;
else
  instruction2  ;
```

Le `test_booleen` est VRAI ou FAUX. S'il est VRAI, l'`instruction1` est exécutée, s'il est FAUX, c'est l'`intruction2`. Il peut être :

- une valeur booléenne `TRUE` ou `FALSE` : `if ($trouve)`
- un calcul retournant un résultat booléen : `if ($age >=65)`
- un calcul booléen suite à une affectation : `if (($c=$i) == 10)`

Aucun point-virgule ne doit terminer la ligne du `if` ou la ligne du `else`.

Exemple

Dans le programme `en_retraite1_shell.php`, si l'âge est supérieur ou égal à 65, le message « En retraite » apparaît, sinon le message « Pas en retraite » est affiché. Le texte « Fin du programme » apparaît systématiquement, il est en dehors du test.

Listing 6.5 – Programme en_retraite1_shell.php

```
<?php
 echo "Entrez votre age : "   ;
 fscanf(STDIN,"%d",$age)      ;
 if ($age >= 65)
   echo "En retraite".PHP_EOL;
 else
   echo "Pas en retraite".PHP_EOL;
 echo "Fin du programme".PHP_EOL;
?>
```

Si le `if` ou le `else` contiennent plusieurs instructions, elles doivent être regroupées en une seule *via* la *séquence* `{}`, comme cela est présenté :

```
if ($age >= 65)
{
  $nban=$age-65 ;
  echo "En Retraite depuis ".$nban." ans".PHP_EOL;
}
else
  echo "Pas en Retraite".PHP_EOL ;
```

- *L'instruction if*

Forme générale

L'instruction `if` (sans `else`) est une version simplifiée où le cas FAUX n'est pas traité. Sa syntaxe est :

```
if (test_booleen)
    instruction1   ;
```

Remarque

PHP distingue cette syntaxe simplifiée de la syntaxe complète en recherchant le mot `else` après l'instruction interne au `if`. Si plusieurs instructions internes au `if` sont présentes et qu'elles ne sont pas regroupées via des accolades, PHP ne trouvant pas de `else` après la première instruction assimilera le test à la version simplifiée, et l'instruction `else` ne sera rattachée à aucun `if`, provoquant une erreur de syntaxe.

Exemple

Les syntaxes suivantes présentent cette forme. Le texte « Au revoir » apparaît toujours, que le message « En retraite » soit affiché ou non.

```
if ($age >= 65)
   echo "En retraite".PHP_EOL;
echo "Au revoir".PHP_EOL     ;
```

- *L'imbrication*

L'imbrication des tests intervient quand une solution possède plus de deux cas. Il est nécessaire d'utiliser un `if…else` comme instruction interne d'un autre.

***if…else* imbriqué**

Prenons le cas de l'embauche d'un salarié et de la vérification préalable de son âge. Trois cas se présentent : il n'a pas encore 16 ans (14 ans pour les apprentis) ; il a 65 ans ou plus ; il a entre 16 et 65 ans (figure 6.3).

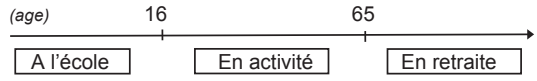


Figure 6.3 – Imbrication de tests.

Le premier test vérifie si l'âge est supérieur ou égal à 65 ans. Si c'est vrai, le message « En retraite » est affiché. Sinon, il faut vérifier si l'âge est inférieur ou supérieur à 16, donc faire un nouveau test dans le `else` du premier.

Le second test vérifie que l'âge est strictement inférieur à 16 ans. Si c'est vrai, on affiche le message « A l'école », sinon on affiche le message « En activité » car il ne reste que le cas où l'âge est compris entre 16 ans (inclus) et 65 ans (exclu).

```
if ($age >= 65)
    echo "En retraite".PHP_EOL    ;
else
    if ($age < 16)
       echo "A l'école".PHP_EOL   ;
    else
       echo "En activité".PHP_EOL ;
```

Remarques

Le `else` du premier `if` ne contient bien qu'une seule instruction qui est un autre `if…else`. Dans ce cas, le recours à la séquence `{}` n'est pas obligatoire.

else if ou *elseif*

Pour tester *N* cas, il faut imbriquer *N*-1 `if…else`, ce qui peut devenir difficile à lire. Une syntaxe particulière améliore la lisibilité de l'imbrication. Les syntaxes précédentes peuvent s'écrire avec `else…if` sur la même ligne :

```
if ($age >= 65)
    echo "En retraite".PHP_EOL ;
else if ($age < 16)
    echo "A l'école".PHP_EOL   ;
else
    echo "En activité".PHP_EOL ;
```

Ou bien encore avec la syntaxe de `elseif` :

```
if ($age >= 65)
    echo "En retraite".PHP_EOL ;
elseif ($age < 16)
    echo "A l'école".PHP_EOL   ;
else
    echo "En activité".PHP_EOL ;
```

else if et *if* imbriqués

Lorsque des instructions `if…else` et `if` sont imbriquées, le déficit de `else` pose un problème d'interprétation. PHP rattache alors le `else` au `if` le plus proche. Dans l'exemple suivant, pour éviter de rattacher le `else` au `if` interne, on utilise la séquence d'instruction qui « isole » le `if` interne (sans le `else`).

```
if ($age >= 16)
{
  if ($age >= 65)
  echo "En retraite".PHP_EOL  ;
}
else
  echo "A l'école".PHP_EOL ;
```

Mettez des accolades, même avec une seule instruction. La syntaxe sera toujours juste et non ambiguë et l'ajout de nouvelles d'instructions n'engendrera aucune erreur.

- *Forme alternative*

Comme cela a été présenté précédemment, il est possible d'utiliser une syntaxe alternative sans les accolades :

- la ligne commençant par `if` est terminée par « `:` » ;
- la ligne commençant par `else`, `else if` ou `elseif` est terminée par « `:` » ;
- l'accolade fermante « } » est remplacée par « `endif ;` ».

```
if (test_booleen) :
  instruction1  ;
  instruction2  ;
else :
  instruction3  ;
  instruction4  ;
endif ;
```

Opérateur conditionnel ou ternaire ? :

- *Forme générale*

Cet opérateur se comporte comme une instruction `if…else`. Sa forme générale est :

```
$variable = test ? expression1 : expression2
```

où :

- `test` : est une expression logique ayant la valeur VRAI ou FAUX ;
- `expression1` : est la valeur retournée, après son évaluation, si test est VRAI ;
- `expression2` : est la valeur retournée, après son évaluation, si test est FAUX.

- *Forme particulière*

Depuis PHP 5.3, `expression1` peut être omise. Sa forme devient :

```
$variable = test ?: expression2
```

Si le test est vrai, la valeur retournée est celle du test, sinon c'est `expression2`.

L'opérateur ternaire est une expression, et n'est pas évalué en tant que variable. Il est recommandé de ne pas imbriquer des expressions ternaires.

- *Exemple*

La syntaxe suivante calcule la valeur absolue de la variable `$i` :

```
$k = (($i>0) ? $i : (-$i)) ;
```

L'instruction *switch*

C'est une instruction conditionnelle « multichoix ». Elle peut remplacer de nombreux cas d'imbrication de `if…else`.

- *Forme générale*

Sa forme générale est :

```
switch (sélecteur)
{
  case expression1 :
       instruction1 ;
       instruction2 ;
       break ;
  case expression2 : case expression3 :
       instruction3 ;
       break ;
  default :
       instruction4 ;
       break ;
}
```

où `selecteur` est :

- une variable booléenne, entière, réelle ou chaîne de caractères : `switch($age)` ;
- un calcul booléen, entier, réel ou chaîne de caractères : `switch($annee%12)`
- `expression1, expression2,…` sont :
  - des constantes de même type que le sélecteur : `1 , 3.5 , "reservation"` ;
  - une expression du même type que le sélecteur : `($age < 65)`.

Le texte « `case:` » est suivi d'une valeur constante (ou d'un calcul) du même type que le sélecteur. Le branchement se fait sur le `case` dont la valeur est celle du sélecteur. Les instructions qui suivent s'exécutent jusqu'à l'instruction `break` qui termine le cas et fait sortir du `switch`. L'absence de `break` continue l'exécution des instructions suivantes, jusqu'au prochain `break`. Regrouper plusieurs cas consiste à les énumérer les uns après les autres, puis à écrire leurs instructions et le `break`.

Le mot `default` est l'équivalent de `else`. Il est facultatif. Si aucun cas n'est sélectionné, le branchement se fait sur le cas `default` s'il est présent.

Remarque

Le caractère « ; » peut remplacer le caractère « : » dans la syntaxe du case.

- *Exemples selon le sélecteur*

Cette section présente l'instruction `switch` avec différents types du sélecteur.

Sélecteur entier

Le programme `switch_entier_shell.php` utilise un sélecteur **entier**. Les valeurs de `$choix` provoquant un traitement sont : 1, 2 et 3. Les valeurs 2 et 3 sont regroupées dans un même traitement. Les valeurs différentes de 1, 2 et 3 sélectionnent le cas `default` qui affiche un message d'erreur.

Listing 6.6 – Programme switch_entier_shell.php

```
<?php
  echo "Entrez votre choix (1, 2 ou 3) : ";
  fscanf(STDIN,"%d",$choix);
  switch ($choix)
  { case 1 : echo "Horaires".PHP_EOL;
             break ;
    case 2 : case 3 : echo "Réservation & Tarifs".PHP_EOL;
             break ;
    default: echo "Choix Impossible".PHP_EOL;
             break ;
  }
?>
```

Voici son exécution :

Listing 6.7 – Exécution de switch_entier_shell.php

```
$ php switch_entier_shell.php
Entrez votre choix (1, 2 ou 3) : 2
Réservation & Tarifs
```

Sélecteur chaîne de caractères

Le programme `switch_chaine_shell.php` utilise un sélecteur **chaîne de caractères**. Les valeurs de `$choix` effectuant un traitement sont : `'reservation'`, `'horaires'` et `'tarifs'`. Les autres valeurs sélectionnent le cas `default`.

Listing 6.8 – Programme switch_chaine_shell.php

```
<?php
  echo "Votre choix (reservations,horaires ou tarifs):";
  fscanf(STDIN,"%s",$choix);
  switch ($choix)
  { case 'reservations' : echo "Réservation".PHP_EOL; break ;
    case 'horaires'     : echo "Horaires".PHP_EOL   ; break ;
    case 'tarifs'       : echo "Tarifs".PHP_EOL     ; break ;
    default: echo "Choix Impossible".PHP_EOL        ; break ;
  }
?>
```

Voici son exécution :

Listing 6.9 – Exécution de switch_chaine_shell.php

```
$ php switch_chaine_shell.php
Votre choix (reservations,horaires ou tarifs): horaires
Horaires
```

Sélecteur réel

Le programme `switch_reel_shell.php` utilise un sélecteur *réel*.

Les valeurs de `$coeff` effectuant un traitement sont : `2.5` et `3.5`.

Listing 6.10 – Programme switch_reel_shell.php

```
<?php
 echo "Entrez une note : ";
 fscanf(STDIN,"%f",$note);
 echo "Entrez un coefficient (ex: 2.5 ou 3.5) : ";
 fscanf(STDIN,"%f",$coeff);
 switch ($coeff)
 {case 2.5 : echo "Mathématiques = ".$note*$coeff.PHP_EOL ;
             break ;
  case 3.5 : echo "Physique      = ".$note*$coeff.PHP_EOL ;
             break ;
  default: echo "Aucune matière pour ce coefficient".PHP_EOL;
           break ;
 }
?>
```

Voici son exécution :

Listing 6.11 – Exécution de switch_reel_shell.php

```
$ php switch_reel_shell.php
Entrez une note : 12.5
Entrez un coefficient (ex: 2.5 ou 3.5) : 3.5
Physique      = 43.75
```

Sélecteur booléen et case expression

Le programme `switch_booleen_interval_shell.php` présente une syntaxe avec un sélecteur *booléen* et des `case` basés sur un calcul booléen.

Listing 6.12 – Programme switch_booleen_interval_shell.php

```
<?php
 echo "Entrez votre âge : "   ;
 fscanf(STDIN,"%d",$age)      ;
 switch (true)
```

```
 { case ((($age >= 65) and ($age < 120)) :
      $nban=$age-65 ;
      echo "En Retraite depuis ".$nban." ans".PHP_EOL;
     break ;
   case ((4 < $age) and ($age < 16)) :
      echo "A l'école".PHP_EOL   ;
      break ;
   case ((16 <= $age) and ($age < 65)) :
      $nban=65-$age ;
      echo "En Activité encore pendant ".$nban." ans".PHP_EOL;
      break ;
   default: echo "Age = ".$age." est invalide".PHP_EOL ;
            break ;
 }
?>
```

Voici son exécution :

Listing 6.13 – Exécution de switch_booleen_interval_shell.php

```
$ php switch_booleen_interval_shell.php
Entrez votre âge : 40
En Activité encore pendant 25 ans
```

Les `case` peuvent utiliser des fonctions booléennes, comme dans le programme `switch_booleen_fonction_shell.php`.

Listing 6.14 – Programme switch_booleen_fonction_shell.php

```
<?php
 echo "Entrez votre note (entière) : " ;
 fscanf(STDIN,"%d",$note)              ;
 switch (true)
 {
   case in_array($note, range(0,10)):
      echo $note." est un résultat faible".PHP_EOL ; break ;
   case in_array($note, range(11,20)):
      echo $note." est un bon résultat".PHP_EOL    ; break ;
   default: echo "Note = ".$note."  invalide ! ".PHP_EOL ;
            break ;
 }
?>
```

- *Forme alternative*

Comme cela a été présenté avec la séquence d'instructions, il est possible d'utiliser une syntaxe alternative sans les accolades.

- La ligne d'instruction commençant par `switch` est terminée par « `:` ».
- L'accolade fermante « } » est remplacée par « `endswitch ;` ».

Voici cette forme alternative :

```
switch (sélecteur) :
  case expression1 :
    instruction1 ;
    instruction2 ;
    break ;
  case expression2 : case expression3 :
    instruction3 ;
    break ;
  default :
    instruction4 ;
    break ;
endswitch ;
```

Les instructions répétitives ou boucles

Présentation

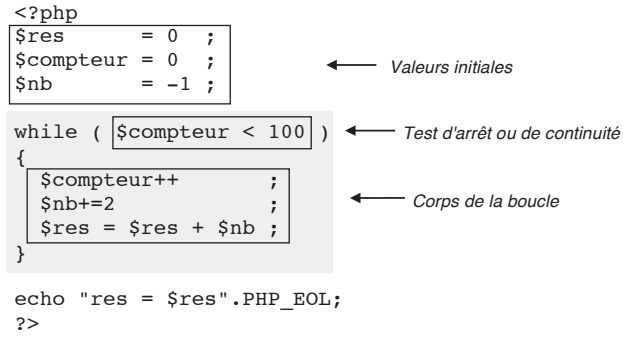
Les boucles représentent le mécanisme le plus important en programmation. C'est aussi le plus difficile à apprendre. Une boucle exécute un certain nombre de fois un groupe d'instructions. Si ce nombre est connu à l'avance, la boucle est dite *déterministe*. Si le nombre d'itérations est *a priori* inconnu, elle est dite *non déterministe*. Les boucles déterministes sont les boucles `for` et `foreach`. Les boucles non déterministes sont les boucles `while` et `do…while`.

La structure d'une boucle

La boucle est constituée de trois parties : le corps de la boucle, les valeurs initiales, le test d'arrêt (ou de continuité). Chaque partie a un rôle précis et possède ses règles de « bon » fonctionnement :

- Le *corps de la boucle* est constitué des instructions internes. Elles implémentent le calcul qui se répète à chaque itération et indiquent *comment passer de l'étape N à l'étape N + 1*. C'est le premier élément de la boucle à construire.
- Les *valeurs initiales* sont les valeurs de départ des variables internes à la boucle. Elles sont définies en dehors de la boucle. Elles doivent être choisies selon le résultat attendu à la fin de la première itération.
- Le *test d'arrêt* doit permettre de sortir de la boucle quand le résultat final est obtenu. Pour valider ce test et éviter une boucle infinie, la variable utilisée dans l'expression booléenne ou comme compteur doit converger vers la valeur finale. Cette variable doit être modifiée dans le corps de la boucle afin d'évoluer.

La figure 6.4 présente les différentes parties d'une boucle `while` qui fait la somme des cent premiers nombres impairs.




```
<?php
$res      = 0  ;
$compteur = 0  ;
$nb       = -1 ;
while ( $compteur < 100 )
{
  $compteur++       ;
  $nb+=2            ;
  $res = $res + $nb ;
}
echo "res = $res".PHP_EOL;
?>
```

Figure 6.4 – La structure d'une boucle.

La boucle while

L'instruction `while` correspond à la boucle TANT QUE. Elle utilise un test booléen de **continuité** (et non d'arrêt). La boucle « tourne » tant que l'expression booléenne est vraie, et s'arrête quand l'expression devient fausse.

- *Forme générale*

La forme générale de l'instruction `while` est :

```
while (test_booleen)
     instruction1 ;
```

Une seule instruction est autorisée. La séquence `{}` doit être utilisée pour en regrouper plusieurs.

```
while (test_booleen)
{
     instruction1 ;
     instruction2 ;
}
```

où `test_booleen` est :

- une valeur ou une variable booléenne VRAI ou FAUX : `while (!$trouve);`
- un calcul logique : `while ($nb < 40);`
- un calcul logique suite à une affectation : `while (($c=$i) != 10).`

De plus `test_booleen` est un test de *continuité* :

- tant qu'il est VRAI, la boucle **continue** ;
- quand il est FAUX, la boucle **s'arrête**.

Il n'y a aucun point-virgule (;) à la fin de la ligne de l'instruction while. Cela produirait une boucle qui exécute l'instruction vide. Si la variable utilisée dans le test de continuité n'est jamais modifiée, la boucle est infinie ! Si le test est FAUX dès le début, la boucle n'est pas exécutée !

- *Forme alternative*

Comme cela a été présenté avec la séquence d'instructions, il est possible d'utiliser une syntaxe alternative sans les accolades.

- La ligne d'instruction commençant par `while` est terminée par « `:` »
- L'accolade fermante « `}` » est remplacée par `endwhile ;`

Voici cette forme alternative :

```
while (test_booleen) :
     instruction1 ;
     instruction2 ;
endwhile ;
```

- *Exemples*

Le programme `while_exemple1_shell.php` affiche les valeurs de 10 à 1.

Listing 6.15 – Programme while_exemple1_shell.php

```
<?php
  $i=10  ;
  while ($i > 0)
     echo $i--.PHP_EOL;
?>
```

Cette boucle `while` ne contient qu'une seule instruction. La variable `$i` régresse dans la boucle (`$i--`). La boucle continue tant que `$i` est supérieure à 0.

Voici son exécution (certains affichages sont remplacés par des pointillés) :

Listing 6.16 – Exécution de while_exemple1_shell.php

```
$ php while_exemple1_shell.php
10
...
2
1
```

Voici la boucle précédente avec deux instructions regroupées par les accolades.

```
$i=10  ;
while ($i > 0)
{
  echo $i.PHP_EOL;
  $i--;
}
```

Remarque

Sans les accolades, la boucle précédente devient infinie. En effet, seul l'affichage par echo serait dans la boucle, la valeur de `$i` ne changerait jamais.

Les lignes suivantes présentent la forme alternative de cette boucle.

```
$i=10  ;
while ($i > 0) :
  echo $i.PHP_EOL;
  $i--;
endwhile;
```

- *Comprendre le principe des boucles*

Principe

La construction d'une boucle suppose souvent d'ajuster les valeurs initiales et le test booléen en la faisant *tourner à la main*. Cette opération consiste à l'exécuter à la place de l'ordinateur, et à noter à chaque itération les valeurs des variables. Ce mécanisme est très utile pour comprendre le dysfonctionnement d'une instruction répétitive, lors d'un développement. Il permet également de comprendre la logique (algorithme) d'un programme.

Comprendre un algorithme

Essayons de comprendre l'algorithme sous-jacent au programme `while_algo_a_trouver_shell.php` en le faisant *tourner à la main*.

Listing 6.17 – Programme while_algo_a_trouver_shell.php

```
<?php
  echo "Entrez 2 nombres entiers : ";
  fscanf(STDIN,"%d %d",$i,$j);
  $cpt=0  ;
  while ($j != 0)
  { $k = $i % $j;
    $i = $j     ;
    $j = $k     ;
    $cpt++      ;
  }
  echo "Le résultat est : ",$i,PHP_EOL ;
?>
```

Supposons que les valeurs saisies pour `$i` et `$j` soient 24 et 42. La figure 6.5 présente les valeurs de `$i`, `$j`, `$k` et `$cpt` pour les différentes itérations (n°) de la boucle. L'itération n° 0 correspond aux valeurs initiales, avant le début de la boucle.

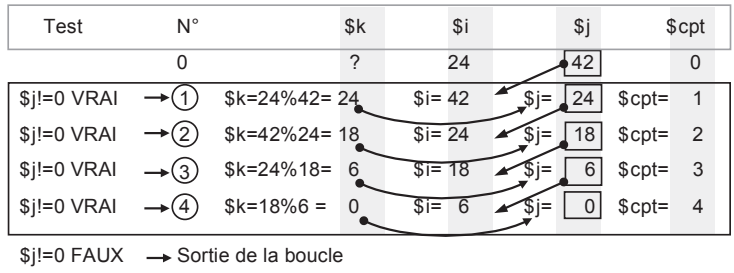


Figure 6.5 – Le déroulement d'une boucle.

À l'itération n° 0, les valeurs de `$i` et `$j` sont 24 et 42. `$cpt` vaut 0 et `$k` est indéfini. `$j` étant différent de 0, on entre dans la boucle.

À l'itération n° 1, `$k` vaut 24, `$i` possède la valeur 42 et `$j` la valeur 24. `$cpt` passe à 1. Cette étape inverse les valeurs de `$i` et de `$j`. `$j` étant différent de 0, on boucle une nouvelle fois.

À l'itération n° 2, `$k` vaut 18, `$i` possède la valeur 24 et `$j` la valeur 18. `$cpt` passe à 2. `$j` étant différent de 0, on boucle une nouvelle fois.

À l'itération n° 3, `$k` vaut 6, `$i` possède la valeur 18 et `$j` la valeur 6. `$cpt` passe à 3. `$j` étant différent de 0, on boucle une nouvelle fois.

À l'itération n° 4, `$k` vaut 0, `$i` possède la valeur 6 et `$j` la valeur 0. `$cpt` passe à 4. `$j` étant égal à 0, on sort de la boucle.

Le résultat final est contenu dans la variable `$i`. Pour les valeurs 24 et 42, le résultat est 6. Les valeurs 24 et 30 auraient donné le même résultat. Les valeurs 24 et 36 donnent le résultat 12. Tous ces résultats sont les plus grands diviseurs (ou dénominateurs) communs des valeurs initiales. Ce programme calcule le PGCD. La variable `$cpt` ne participe pas au calcul, elle indique juste le nombre d'itérations.

Faire tourner une boucle à la main est le seul moyen efficace pour comprendre son fonctionnement et la corriger. Pour être efficace, il ne faut pas « réfléchir » au sens des opérations, mais appliquer « mécaniquement » les instructions. Une fois les calculs effectués, on peut comparer leurs résultats avec les valeurs espérées à chaque itération, ou à la fin de la boucle. Les différences permettent de trouver les instructions défaillantes.

Construire une boucle

L'étape suivante consiste à construire une boucle devant résoudre un problème donné. Elle se fera en trois étapes :

1. Constitution du **corps de la boucle**.
2. Définition des **valeurs initiales**.
3. Écriture du **test d'arrêt** (ou de continuité).

Prenons l'exemple d'un programme qui doit calculer x^m. Il faut préalablement réécrire ce calcul sous une forme itérative : $x^m = x*x*x*\ldots*x$, m fois, pour mettre en évidence le calcul qui se répète : la multiplication par x.

Constitution du corps de la boucle – Trouver les instructions internes à la boucle revient à pointer le calcul qui fait passer de l'étape N à l'étape $N + 1$. Le résultat de l'étape N est supposé connu, c'est l'*ancien* résultat. Le résultat de l'étape $N + 1$ est le *nouveau* résultat. Celui-ci servira de point de départ au calcul suivant ($N + 2$), et sera l'ancien résultat pour l'étape $N + 2$. C'est donc la même variable qui contient l'ancien et le nouveau résultat. À droite du signe d'affectation se trouve la valeur précédente qu'on lit, à gauche se trouve la nouvelle valeur, qu'on affecte. La figure 6.6 présente le calcul de l'étape $N + 1$.

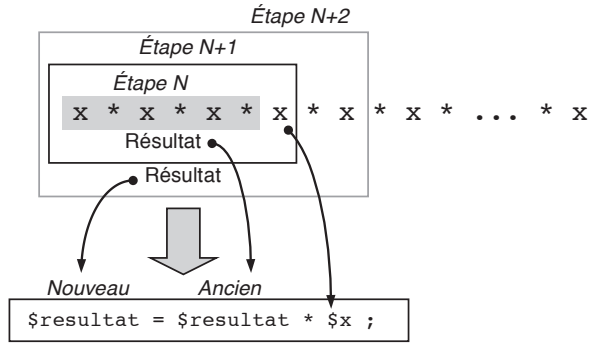


Figure 6.6 – Le corps de la boucle.

Le corps de la boucle est constitué de l'instruction suivante :

```
$resultat = $resultat * $x ;
```

La boucle doit être effectuée « *m* » fois. Il faut ajouter une nouvelle instruction comptant le nombre d'itérations. Le nouveau corps est le suivant :

```
$compteur++ ;
$resultat = $resultat * $x ;
```

Définition des valeurs initiales – Les variables internes à la boucle, situées à droite d'une affectation, doivent posséder une *valeur de départ*. La valeur initiale de `$resultat` doit aboutir à la fin du premier passage au résultat x^1, soit x. Pour que `$resultat*$x` donne `$x`, la valeur initiale de `$resultat` doit être égale à 1. De même, `$compteur` doit être égal à 1 à la fin de la première itération, sa valeur initiale doit être égale à 0.

```
$compteur = 0 ;
$resultat = 1 ;
while (   )
{
  $compteur++ ;
  $resultat = $resultat * $x ;
}
```

Écriture du test d'arrêt (ou de continuité) – Le test doit limiter l'exécution de la boucle à « *m* » fois. Il suffit de poser un test « *a priori* », et de le vérifier en faisant tourner la boucle. `$compteur` indique le nombre d'itérations ; il sert de test d'arrêt. Posons le test suivant :

```
while ($compteur <= $m)
```

En faisant tourner la boucle, on s'aperçoit que ce test est FAUX. Il effectue un calcul de trop ! Prenons le calcul de x^3 ; la valeur de « *m* » est 3.

Test d'entrée 1er passage : compteur vaut 0, compteur <= 3 est VRAI. À la fin du 1er passage, compteur = 1 et résultat vaut x.

Test d'entrée 2e passage : compteur vaut 1, compteur <= 3 est VRAI. À la fin du 2e passage, compteur = 2 et résultat vaut x^2.

Test d'entrée 3e passage : compteur vaut 2, compteur <= 3 est VRAI. À la fin du 3e passage, compteur = 3 et résultat vaut x^3.

Test d'entrée 4e passage : compteur vaut 3, compteur <= 3 est VRAI. La boucle s'exécute une nouvelle fois alors qu'elle devrait s'arrêter. La valeur 3 ne doit pas permettre une nouvelle exécution. Le test doit être corrigé en strictement inférieur :

```
while ($compteur < $m)
```

Voici le programme complet `while_puissance_shell.php`.

Listing 6.18 – Programme while_puissance_shell.php

```
<?php
 echo "Entrez un nombre réel (ex: 2.3): ";
 fscanf(STDIN,"%f",$x);
 echo "Entrez la puissance entière (ex: 3) : ";
 fscanf(STDIN,"%d",$m);
 $compteur = 0 ;
 $resultat = 1 ;
 while ($compteur < $m)
 {
   $compteur++ ;
   $resultat = $resultat * $x ;
 }
 echo "$x ** $m = $resultat".PHP_EOL;
?>
```

Voici son exécution :

Listing 6.19 – Exécution de while_puissance_shell.php

```
$ php while_puissance_shell.php
Entrez un nombre réel (ex: 2.3): 2.5
Entrez la puissance entière (ex: 3) : 5
2.5 ** 5 = 97.65625
```

- *Règles à respecter sur les boucles*

Les boucles mal construites peuvent ne jamais s'exécuter, ou bien ne jamais s'arrêter (infinie). Voici quelques règles à respecter :

- Pour que la boucle démarre il faut de *bonnes valeurs initiales*. La boucle suivante s'exécute 0 fois car le test de continuité est faux dès le début !

```
$i = 10 ;
while ($i < 0) // $i est positif: test faux dès le début
{ $i-- ;
```

```
  echo $i.PHP_EOL ;
}
```

- La variable utilisée dans le test d'arrêt (ou de continuité) *doit toujours évoluer* dans la boucle. La boucle suivante est infinie, car la variable utilisée dans le test de continuité n'est jamais modifiée dans la boucle. Le test est toujours vrai !

```
$i = -1 ;
while ($i < 0) // boucle infinie: $i jamais modifié
  echo $i.PHP_EOL ;
```

- La variable de test doit *converger vers la valeur finale*. La boucle suivante est infinie. La variable `$i` s'éloigne de la valeur finale !

```
$i = -10 ;
while ($i < 0)
{ $i-- ; // $i s'éloigne de 0: test toujours vrai
  echo $i.PHP_EOL ;
}
```

- Le critère d'arrêt *doit être atteint* par la variable de test. La boucle suivante est infinie. La variable `$i` converge vers la valeur finale mais la dépasse pour ensuite diverger ! Il faut éviter les tests d'égalité ou de différence `($i != 0)` et préférer les tests comme `($i <= 0)` quand cela est possible.

```
$i = -5 ;
while ($i != 0)
{ $i+=2 ;        // passe "par dessus" le test : -1 puis +1
  echo $i.PHP_EOL ;
}
```

- Faire « *tourner à la main* » le programme pour vérifier qu'il correspond bien à l'algorithme désiré.

La boucle do…while

L'instruction `do…while` est non déterministe. C'est la boucle FAIRE…TANT QUE. Elle exécute les instructions internes tant que le test est VRAI.

- *Forme générale*

La forme générale de l'instruction est :

```
do
  instruction1  ;
while (test_booleen) ;
```

où `test_booleen` est :

- une valeur ou une variable booléenne VRAI ou FAUX : `while (!$trouve)` ;
- un calcul logique : `while ($nb < 40)` ;
- un calcul logique suite à une affectation : `while (($c=$i) != 10)`.

De plus `test_booleen` est un test de *continuité* :

- tant qu'il est VRAI, la boucle *continue* ;
- quand il est FAUX, la boucle *s'arrête*.

La boucle `do…while` ne contient qu'une seule instruction. Une séquence `{}` doit être utilisée pour en regrouper plusieurs. Cette boucle diffère de la boucle `while` par son test qui est évalué à la fin, elle sera donc toujours effectuée au moins une fois ! La ligne contenant le mot `while` est la dernière ligne de l'instruction, elle est terminée par un point-virgule. Il n'y a pas de point-virgule terminant la ligne `do`.

- *Exemple*

Le programme `do_while1_shell.php` affiche la multiplication par 3 du nombre saisi tant que la valeur 0 n'est pas entrée.

Listing 6.20 – Programme do_while1_shell.php

```
<?php
 $base=3;
 do
 {
  echo "Entrez un nombre (0=FIN) : " ;
  fscanf(STDIN,"%d",$nb);
  echo $nb." x ".$base." = ".$nb*$base.PHP_EOL;
 }
 while ($nb != 0);
?>
```

La boucle *for*

L'instruction `for` est la boucle POUR. C'est une boucle déterministe. Elle doit être utilisée *quand le nombre d'itérations est connu à l'avance*. Elle utilise une variable qui sert de compteur de boucle.

- *Forme générale*

Sa forme générale est :

```
for (expression1 ; test_booleen ; expression2)
      instruction1  ;
```

Une seule instruction est autorisée. La séquence `{}` doit être utilisée pour en regrouper plusieurs.

```
for (expression1 ; test_booleen ; expression2)
{
     instruction1 ;
     instruction2 ;
}
```

où :

- `expression1` est la définition de la valeur initiale : `for (**$i=0**;...;...)`
- `test_booleen` est un calcul logique : `for ($i=0;**$i<10**;...)`
- `expression2` est la modification du compteur : `for ($i=0;$i<10;**$i++**)`

La variable utilisée dans l'en-tête de la boucle `for` est appelée compteur de boucle. Elle peut être de type entier, chaîne de caractères ou réel.

Il n'y a pas de point-virgule (;) à la fin de la ligne de l'instruction for. Cela aurait pour effet de créer une boucle qui exécuterait l'instruction vide. La variable utilisée comme compteur de boucle ne doit jamais être modifiée dans le corps de la boucle. Le résultat est imprévisible !

- *Forme alternative*

Comme cela a été présenté avec la séquence d'instructions, il est possible d'utiliser une syntaxe alternative sans les accolades.

- La ligne d'instruction commençant par `for` est terminée par « : »
- L'accolade fermante « } » est remplacée par `endfor ;`

  Voici cette forme alternative :

```
for (expression1;test_booleen;expression2) :
     instruction1 ;
     instruction2 ;
endfor ;
```

- *Exemples*

Cette section présente des boucles `for` avec différents types de compteur.

Avec un compteur entier

Le programme `for_exemple_entier_shell.php` affiche la table de multiplication indiquée par l'utilisateur.

Listing 6.21 – Programme for_exemple_entier_shell.php

```
<?php
 echo "Entrez une base de calcul : " ;
 fscanf(STDIN,"%d",$base);
 for ($i=1 ; $i<11 ; $i++)
 { $res = $i * $base ;
   echo "$i x $base = $res".PHP_EOL;
 }
?>
```

Voici son exécution :

Listing 6.22 – Exécution de for_exemple_entier_shell.php

```
$ php for_exemple_entier_shell.php
Entrez une base de calcul : 3
1 x 3 = 3
2 x 3 = 6
...
10 x 3 = 30
```

Avec un compteur chaîne de caractères

Le programme `for_exemple_chaine_shell.php` présente une boucle avec un compteur de type **chaîne de caractères**.

Listing 6.23 – Programme for_exemple_chaine_shell.php

```
<?php
for ($lettre = 'A'; $lettre != 'AA' ; $lettre++)
     echo "$lettre ";
echo PHP_EOL;
?>
```

Voici son exécution :

Listing 6.24 – Exécution de for_exemple_chaine_shell.php

```
$ php for_exemple_chaine_shell.php
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Attention avec les compteurs de type chaînes de caractères. En effet la progression avec `$lettre++` peut amener à des chaînes plus longues selon le test.

La boucle du programme `for_exemple_chaine2_shell.php` parcourt les chaînes de «A» à «YZ» à cause du test `$lettre <='Z'`:

Listing 6.25 – Programme for_exemple_chaine2_shell.php

```
<?php
$compteur=0;
for ($lettre = 'A'; $lettre <= 'Z' ; $lettre++)
{ echo "$lettre ";
  $compteur++;
  if ($compteur == 26)
  {echo PHP_EOL;
   $compteur=0;
  }
}
echo PHP_EOL;
?>
```

Voici son exécution (certaines lignes sont remplacées par des « … ») :

Listing 6.26 – Exécution de for_exemple_chaine2_shell.php

```
$ php for_exemple_chaine2_shell.php
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
AA AB AC AD AE AF AG AH AI AJ AK AL AM AN AO AP AQ AR AS AT AU AV AW AX
AY AZ
...
YA YB YC YD YE YF YG YH YI YJ YK YL YM YN YO YP YQ YR YS YT YU YV YW YX
YY YZ
```

Avec un compteur réel

Le programme `for_exemple_reel_shell.php` affiche une suite de réels en fonction d'une valeur de départ, d'une valeur d'arrivée et d'un pas de progression.

Listing 6.27 – Programme for_exemple_reel_shell.php

```
<?php
 // Saisie des données
 echo "Valeur réelle de départ (ex : 2.6) : " ;
 fscanf(STDIN,"%f",$Reel_1)    ;
 echo "Valeur réelle d'arrivée (ex : 3.6) : " ;
 fscanf(STDIN,"%f",$Reel_2)    ;
 echo "Pas de progression      (ex : 0.1) : " ;
 fscanf(STDIN,"%f",$Pas_Progression)    ;
 for ($Reel=$Reel_1;$Reel<=$Reel_2;$Reel+=$Pas_Progression)
 { echo "$Reel\t";}
 echo PHP_EOL;
?>
```

Voici son exécution :

Listing 6.28 – Exécution de for_exemple_reel_shell.php

```
$ php for_exemple_reel_shell.php
Valeur réelle de départ (ex : 2.6) : 2.1
Valeur réelle d'arrivée (ex : 3.6) : 3.3
Pas de progression      (ex : 0.1) : 0.1
2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8  2.9  3  3.1  3.2
```

- *Forme évoluée*

La boucle `for` peut utiliser deux compteurs ou plus. Le programme `for_evolue_shell.php` inverse la chaîne `$nom`, *via* les compteurs `$i` et `$j`.

Listing 6.29 – Programme for_evolue_shell.php

```
<?php
  echo "Entrez un nom : " ;
  fscanf(STDIN,"%s",$nom) ;
  for ( $i=0, $j=strlen($nom)-1 ; $i<$j ; $i++, $j-- )
```

```
  {
    $c       = $nom[$i];
    $nom[$i] = $nom[$j];
    $nom[$j] = $c      ;
  }
  echo "Résultat : $nom".PHP_EOL;
?>
```

Voici son exécution :

Listing 6.30 – Exécution de for_evolue_shell.php

```
$ php for_evolue_shell.php
Entrez un nom : dupont
Résultat : tnopud
```

Le compteur de la boucle for doit être un entier, une chaîne ou un réel. Si la valeur finale est atteinte dès le premier passage, alors la boucle s'exécute zéro fois. À la sortie de la boucle, le compteur possède la dernière valeur affectée par la boucle. La boucle suivante dont l'en-tête ne contient que des « ; » est infinie : `for (;;) { … }`

- *Imbrication des boucles for*

L'imbrication consiste à exécuter une boucle à l'intérieur d'une autre. Cela s'applique à toutes les boucles. L'imbrication des boucles `for` est très utilisée pour la gestion des tableaux à plusieurs dimensions.

Lorsque deux boucles sont imbriquées, la boucle interne « tourne plus vite » que la boucle externe. Pour chaque valeur du compteur de la boucle principale, la seconde boucle est exécutée en totalité. Les instructions les plus internes sont exécutées $N \times M$ fois, où N est le nombre d'itérations de la boucle principale, et M celui de la seconde boucle. Le programme `for_tables_multiplication_shell.php` affiche les tables de multiplications de 1 à 12 :

Listing 6.31 – Programme for_tables_multiplication_shell.php

```
<?php
 for ( $i=1 ; $i<13 ; $i++  ) // Boucle externe
 { echo "--- Table de $i ---".PHP_EOL;
   for ($j=1 ; $j <11 ; $j++) // Boucle interne
   { $res=$i * $j ;
     printf("%4d",$res) ;
   }
   echo PHP_EOL ;
 }
?>
```

Voici son exécution (certaines lignes sont remplacées par des « … ») :

Listing 6.32 – Exécution de for_tables_multiplication_shell.php

```
$ php for_tables_multiplication.php
--- Table de 1 ---
   1   2   3   4   5   6   7   8   9  10
--- Table de 2 ---
   2   4   6   8  10  12  14  16  18  20
...
--- Table de 12 ---
  12  24  36  48  60  72  84  96 108 120
```

La boucle *foreach*

La boucle `foreach` est spécifique aux tableaux. Nous présentons dans cette section sa syntaxe, mais elle sera abordée plus largement, dans le chapitre sur les données structurées. Elle fournit une syntaxe simple pour parcourir les différents éléments d'un tableau et permet de retrouver à la fois *la valeur*, et *l'indice* de chaque élément.

• *Formes générales*

Il existe deux syntaxes de la boucle `foreach`.

Première syntaxe

La première syntaxe récupère *la valeur* de chaque élément.

```
foreach ($var_tableau as $var_contenu)
      instruction1  ;
```

Ou avec plusieurs instructions :

```
foreach ($var_tableau as $var_contenu)
{
     instruction1 ;
     instruction2 ;
}
```

où :

- `$var_tableau` est la variable tableau ;
- `$var_contenu` est la variable recevant la valeur de chaque élément.

Deuxième syntaxe

La deuxième syntaxe récupère *la valeur et l'indice* de chaque élément.

```
foreach ($var_tableau as $indice => $var_contenu)
      instruction1  ;
```

Ou avec plusieurs instructions :

```
foreach ($var_tableau as $indice => $var_contenu)
{
     instruction1 ;
     instruction2 ;
}
```

où :

- `$var_tableau` est la variable tableau ;
- `$indice` est la variable recevant l'indice de chaque élément ;
- `$var_contenu` est la variable recevant la valeur de chaque élément.

- *Forme alternative*

Comme cela a été présenté avec la séquence d'instructions, il est possible d'utiliser une syntaxe alternative sans les accolades.

- La ligne d'instruction commençant par `foreach` est terminée par « `:` ».
- L'accolade fermante « } » est remplacée par `endforeach ;`.

  Voici cette forme alternative :

```
foreach ($var_tableau as $var_contenu) :
     instruction1 ;
     instruction2 ;
endforeach ;
```

Ou bien, avec récupération de chaque indice :

```
foreach ($var_tableau as $indice => $var_contenu) :
     instruction1 ;
     instruction2 ;
endforeach ;
```

- *Exemples*

Le programme `foreach_exemple_tabentiers_shell.php` affiche le contenu d'un tableau d'entier.

Listing 6.33 – Programme foreach_exemple_tabentiers_shell.php

```
<?php
// -- Création du tableau avec les valeurs 10, 20, 30 et 40 --
$tab = array(10, 20, 30, 40);
echo "--foreach: contenu, tableau d'entiers--".PHP_EOL;
// -- Boucle d'affichage --
foreach ($tab as $valeur)
{ echo '$valeur='.$valeur.PHP_EOL; }
echo "--foreach: contenu,indices tableau d'entiers--".PHP_EOL;
foreach ($tab as $indice => $valeur)
{echo '$indice = '.$indice.'  $valeur='.$valeur.PHP_EOL; }
```

```
// -- Boucle de calcul --
echo "--Calcul (x2) sur le tableau d'entiers--".PHP_EOL;
foreach ($tab as $valeur)
{ $res = $valeur * 2;
  echo '$res='.$res.PHP_EOL;
}
?>
```

Voici son exécution :

Listing 6.34 – Exécution de foreach_exemple_tabentiers_shell.php

```
$ php foreach_exemple_tabentiers.php
--foreach: contenu, tableau d'entiers--
$valeur=10
$valeur=20
$valeur=30
$valeur=40
--foreach: contenu,indices tableau d'entiers--
$indice = 0  $valeur=10
$indice = 1  $valeur=20
$indice = 2  $valeur=30
$indice = 3  $valeur=40
--Calcul (x2) sur le tableau d'entiers--
$res=20
$res=40
$res=60
$res=80
```

- *Imbrication des boucles foreach*

Dans le cas de tableaux à *N* dimensions, il est nécessaire d'utiliser plusieurs boucles (*N* boucles) `foreach` imbriquées.

Imbrication de deux boucles

Pour un tableau à deux dimensions, la première boucle récupère la ligne, la seconde boucle (interne), chaque élément de cette ligne.

Le programme `foreach_exemple_imbrication2D_shell.php` affiche les informations d'un tableau à deux dimensions d'un crédit. Chaque ligne contient les informations d'une échéance. La première colonne indique la mensualité et la deuxième colonne la valeur de l'amortissement.

Tableau 6.4 – Tableau des échéances

N° de ligne = échéance	Mensualité	Amortissement
1	2 000	1 000
2	1 900	1 100
3	1 800	1 200

Listing 6.35 – Programme foreach_exemple_imbrication2D_shell.php

```
<?php
// ------------------------------------
// Création du tableau à deux dimensions
// ------------------------------------
$Tab_Amort = array(
   1 => array("Mensualité"=>2000, "Amortissement"=> 1000),
   2 => array("Mensualité"=>1900, "Amortissement"=> 1100),
   3 => array("Mensualité"=>1800, "Amortissement"=> 1200),
   );
// --- foreach imbriquées ---
foreach ($Tab_Amort as $NumEcheance => $Tab_Amort_Ligne)
{
  echo "Echéance = $NumEcheance\t";
  foreach ($Tab_Amort_Ligne as $clef => $valeur)
  {
    echo "$clef = $valeur\t";
  }
  echo PHP_EOL;
}
?>
```

Voici son exécution :

Listing 6.36 – Exécution de foreach_exemple_imbrication2D_shell.php

```
$ php foreach_exemple_imbrication2D_shell.php
Echéance = 1    Mensualité = 2000  Amortissement = 1000
Echéance = 2    Mensualité = 1900  Amortissement = 1100
Echéance = 3    Mensualité = 1800  Amortissement = 1200
```

Imbrication de trois boucles

Le programme `foreach_exemple_imbrication3D_shell.php` affiche les informations d'un tableau d'amortissement à trois dimensions d'un crédit. Chaque ligne correspond aux éléments d'une échéance. La première colonne indique la mensualité et la deuxième colonne la valeur de l'amortissement. La colonne Mensualité est décomposée en deux sous-colonnes « Montant » et « Type ». La colonne Amortissement est décomposée en deux sous-colonnes « Montant » et « Pourcentage ».

Tableau 6.5 – Tableau des échéances 3D

Échéance	Mensualité		Amortissement	
	Montant	Type	Montant	Pourcentage
1	2 000	AssComp	1 000	50
2	1 900	AssComp	1 100	58
3	1 800	AssComp	1 200	66

Listing 6.37 – Programme foreach_exemple_imbrication3D_shell.php

```
<?php
// -------------------------------------
// Création du tableau à trois dimensions
// -------------------------------------
$Tab_Amort3D = array(
 1 => array(
  "Mensualité"=>array("Montant"=>2000,"Type"=>"AssComp"),
  "Amortissement"=>array("Montant"=>1000,"Pourcent"=>50)),
 2 => array(
  "Mensualité"=>array("Montant"=>1900,"Type"=>"AssComp"),
  "Amortissement"=>array("Montant"=>1100,"Pourcent"=>58)),
 3 => array(
  "Mensualité"=>array("Montant"=>1800,"Type"=>"AssComp"),
  "Amortissement"=>array("Montant"=>1200,"Pourcent"=>66)),
   );
// --- foreach imbriquées ---
foreach ($Tab_Amort3D as $NumEcheance => $Tab_Amort_Mens)
{
  echo "Echéance = $NumEcheance\t";
  foreach ($Tab_Amort_Mens as $Type => $Tab_Type)
  {
    foreach ($Tab_Type as $clef => $valeur)
    {
     echo "$clef=$valeur\t";
    }
  }
  echo PHP_EOL;
}
?>
```

Voici son exécution :

Listing 6.38 – Exécution de foreach_exemple_imbrication3D_shell.php

```
$ php foreach_exemple_imbrication3D_shell.php
Echéance=1 Montant=2000 Type=AssComp Montant=1000 Pourcent=50
Echéance=2 Montant=1900 Type=AssComp Montant=1100 Pourcent=58
Echéance=3 Montant=1800 Type=AssComp Montant=1200 Pourcent=66
```

6.4 Les instructions de contrôle

L'instruction *break*

L'instruction break fait sortir d'une instruction composée `switch`, `for`, `foreach`, `while`, `do...while`. Elle a été présentée dans ce chapitre avec l'instruction `switch`. Appliquée aux boucles, elle sort de la boucle et passe à l'instruction située après celle-ci. Sa syntaxe est :

```
break ;
```

ou

```
break 2 ; // break N ;
```

Si une valeur numérique est indiquée en argument, l'instruction `break` fait sortir de *N* niveaux d'imbrication. Dans le programme `break_niveaux_shell.php`, la syntaxe `break 2 ;` fait sortir du `switch` et de la boucle `while`.

Listing 6.39 – Programme break_niveaux_shell.php

```
<?php
$choix=-1;
while ($choix != 2)
{echo "Entrez une valeur 0, 1 ou 2 : ";
 fscanf(STDIN,"%d",$choix);
 switch($choix)
 { case 1 : echo "Choix 1 sélectionné".PHP_EOL ;
            break ; // sortie du switch
   case 0 : echo "Choix 0 sélectionné".PHP_EOL ;
            break 2 ; // sortie du switch et du while
 }
}
echo "Fin du programme".PHP_EOL;
?>
```

Il est toujours préférable de sortir d'une boucle *via* son test d'arrêt qui doit être bien écrit. Si l'on désire sortir d'une boucle `for` par une instruction `break`, c'est que la boucle est non déterministe et qu'il faut la remplacer avec une boucle `while` !

L'instruction *continue*

L'instruction `continue` est utilisée quand on souhaite passer directement à l'itération suivante d'une boucle, sans exécuter les instructions internes qui suivent. Le programme `continue_boucle_shell.php` affiche l'écho de la valeur saisie et termine la boucle lorsque la saisie de la valeur est 2.

Listing 6.40 – Programme continue_boucle_shell.php

```
<?php
$choix=-1;
while ($choix != 2)
{
 echo "Entrez une valeur numérique (2=FIN) : ";
 fscanf(STDIN,"%d",$choix);
 if($choix == 2)
    continue;
 echo "Choix $choix sélectionné".PHP_EOL ;
}
echo "Fin du programme".PHP_EOL;
?>
```

Il faut écrire correctement les tests ou les boucles et ne pas abuser de cette instruction ! Dans cet exemple, il suffit d'écrire `if ($choix !=2)` et l'instruction continue devient inutile.

L'instruction *declare*

Cette instruction définit des directives d'exécution (indiquées à l'interpréteur PHP) sur un ensemble de lignes de programmation. Deux directives sont reconnues, `ticks` et `encoding`.

Ticks

Un tick est un évènement spécifié par la fonction `register_tick_function()` qui intervient toutes les *N* commandes de bas niveau. En voici un exemple :

Listing 6.41 – Programme declare_tick_shell.php

```
<?php
 declare(ticks=5); // toutes les 5 commandes de bas niveau
 // Cette fonction est exécutée à chaque évènement "tickable"
 function fonction_tick()
 {
  global $variable;
  echo "fonction_tick() appelée : variable=$variable".PHP_EOL;
 }
 register_tick_function('fonction_tick');
 $variable = 10;
 if ($variable > 0)
 {$variable *= 2;}
 ?>
```

Encoding

L'encodage (table de codage des caractères) utilisé dans un programme peut être spécifique à ce programme. La syntaxe suivante indique l'usage de l'Iso-Latin1.

```
<?php
declare(encoding='ISO-8859-1');
// les lignes du programme ...
?>
```

Les instructions *include* et *require*

Syntaxe générale

L'instruction `include` a été présentée au chapitre 2. Elle inclut le texte contenu dans un fichier à un endroit précis du programme. Elle permet, par exemple :

- de regrouper dans des fichiers spécifiques des syntaxes redondantes telles que l'entête « header » ou le pied de page « footer » ;

- de regrouper dans un fichier PHP les procédures et les fonctions « outils » développées dans le programme.

La modification des pages ou des fonctions contenues dans le fichier à inclure est directement propagée aux programmes contenant l'inclusion. Sa syntaxe est :

```
include "RepSprog/sous_programmes_communs.php" ;
```

ou

```
include 'RepSprog/sous_programmes_communs.php' ;
```

ou

```
include("site_include_header.php");
```

Le premier exemple et le deuxième exemple indiquent le chemin relatif du sous-répertoire `RepSprog` dans lequel chercher le fichier. Le troisième exemple recherche le fichier dans le répertoire du programme PHP. Si le fichier est absent, une erreur (`E_WARNING`) est émise, mais le programme continue son exécution.

L'instruction `require` effectue la même inclusion. Mais contrairement à `include`, si le fichier est absent, le programme émet une erreur fatale (`E_COMPILE_ERROR`) et s'arrête. Sa syntaxe est similaire :

```
require "RepSprog/sous_programmes_communs.php" ;
```

ou

```
require 'RepSprog/sous_programmes_communs.php' ;
```

ou

```
require("site_include_header.php");
```

La page appelée par l'inclusion doit être une page PHP, donc ayant l'extension .php

Portée des variables

Les instructions internes au fichier inclus héritent des variables connues *au moment* où apparaît cette ligne d'inclusion dans le programme. Contrairement aux variables, les fonctions ou classes ont une portée globale, elles sont visibles au code inclus même si elles apparaissent après la ligne d'inclusion. Si l'inclusion apparaît dans une fonction, le code inclus est considéré comme faisant partie de la fonction.

Contrainte sur le fichier inclus

S'il est possible d'inclure des syntaxes HTML en les écrivant telles quelles dans un fichier à inclure `.php`, les instructions PHP à inclure doivent être placées entre les balises de PHP `<?php` et `?>`.

Valeur retournée

Il est possible de récupérer le résultat de l'exécution de l'instruction `include`. Si le fichier à inclure n'est pas trouvé, la valeur retournée est FALSE et l'erreur `E_WARNING` est émise. Si le fichier est trouvé, la valeur retournée est 1. Si le fichier est trouvé et qu'il contient un `return`, le retour est la valeur du `return`. Le programme `include_appel_shell.php` inclut trois fichiers. Le premier n'existe pas, le deuxième retourne « valeur retournée », le troisième ne retourne rien.

Listing 6.42 – Programme include_appel_shell.php

```
<?php
 $retour1 = include 'include_appele_inexistant_shell.php';
 echo "retour1=$retour1".PHP_EOL; //FALSE et E_WARNING émis
 $retour2 = include 'include_appele_return_shell.php';
 echo "retour2=$retour2".PHP_EOL; //Affiche 'valeur retournée'
 $retour3 = include 'include_appele_pas_de_return_shell.php';
 echo "retour3=$retour3".PHP_EOL;  //Affiche 1
?>
```

Listing 6.43 – Programme include_appele_return_shell.php

```
<?php
 $variable="valeur retournée";
 return $variable;
?>
```

Listing 6.44 – Programme include_appele_pas_de_return_shell.php

```
<?php
 $variable="valeur pas retournée";
?>
```

Voici l'exécution de `include_appel_shell.php` :

Listing 6.45 – Execution de include_appel_shell.php

```
$ php include_appel_shell.php
Warning:include(include_appele_inexistant_shell.php):failed ..
retour1=
retour2=valeur retournée
retour3=1
```

Les instructions *include_once* ou *require_once*

Ces instructions sont similaires à `include`, pour `include_once`, et `require` pour `require_once`. La différence est que si le fichier a déjà été inclus, il ne le sera pas à nouveau. Il faut privilégier ces instructions si plusieurs inclusions d'un fichier risquent d'être effectuées alors qu'on désire ne l'inclure qu'une seule fois.

Exercices

6.1 Faire un programme qui lit trois notes d'examens et trois coefficients (n1, n2, n3 et c1, c2, c3 sont des réels) et qui affiche la moyenne pondérée et la mention. Le traitement de la mention sera effectué avec un `switch`. Voici un exemple d'exécution du programme attendu :

```
$ php switch_note_mention_reel_shell.php
Entrez 3 notes        : 12.5 13.5 14.5
Entrez 3 coefficients : 1.5 2.5 3.5
Votre moyenne est : +13.77
Mention ASSEZ BIEN
```

6.2 Adapter le programme précédent pour effectuer la saisie dans un formulaire HTML en méthode POST, et afficher le résultat dans un page web.

6.3 Faire un programme shell qui simule les Mensualités d'un crédit, pour les Taux d'intérêts compris entre un *taux de départ*, et un *taux final* selon un pas de progression. Les saisies sont le *Capital emprunté*, le *Nombre d'années*, le *Taux annuel de l'assurance*, le *Taux d'intérêt hors assurance de départ*, et le *Taux d'intérêt hors assurance final*. Partir du programme `emprunt_shell.php` en exercice au chapitre 5. La formule de calcul de la mensualité d'un crédit est :

$$MensHA = \text{Cap} \times \text{TauxM} \times \frac{(1 + \text{TauxM})^{\text{NbMois}}}{(1 + \text{TauxM})^{\text{NbMois}} - 1}$$

où :

- *MensHA* est la mensualité hors assurance ;
- *Cap* est le capital emprunté ;
- *TauxM* est le taux mensuel = taux annuel/12 ;
- *NbMois* est le nombre de mois = nombre d'années × 12.

Le calcul du coût final du crédit en % correspond à :

$$PourentCoutTotal = \frac{CoutTotal}{Cap} \times 100$$

Seront affichés comme résultats : la *mensualité*, le *coût mensuel de l'assurance*, le *coût total du crédit* en euros (somme des intérêts mensuels), le *coût total du crédit en %* (rapport du coût total sur le capital emprunté).

Voici un exemple d'exécution pour un crédit de 300 000 € sur 20 ans avec un taux d'assurance de 0,33 %. La simulation porte sur les taux entre 1,8 % et 2,1 % selon un pas de progression de 0,1 %, ce qui simule les mensualités pour les taux 1,8 %, 1,9 %, 2,0 %, 2,1 %. Certains affichages sont remplacés par des « … ».

```
$ php for_simulateur_credit_shell.php
Capital: 300000
Nombre d'années : 20
Taux de l'Assurance (0.29) :0.33
Taux Annuel Hors Assurance le plus bas (ex : 2.6) : 1.8
Taux Annuel Hors Assurance le plus haut(ex : 3.6) : 2.1
Pas de progression du taux (ex : 0.1) : 0.1
--- Simulation avec Taux      = 1.8 ---
Mensualité Assurance Comprise : 1571.9
Coût de l'Assurance par mois  : 82.5
Coût du crédit en €           : 57456
Coût du crédit en %           : 19.15
--- Simulation avec Taux      = 1.9 ---
...
--- Simulation avec Taux      = 2.1 ---
Mensualité Assurance Comprise : 1614.4
Coût de l'Assurance par mois  : 82.5
Coût du crédit en €           : 67656
Coût du crédit en %           : 22.55
```

6.4 Adapter le programme `for_simulateur_credit_shell.php` précédent pour effectuer la saisie dans un formulaire HTML en méthode POST, et afficher le résultat dans un page web sous la forme d'un tableau.

Les figures 6.7 et 6.8 présentent la page de saisie et la page de résultat (feuille de styles optionnelle).

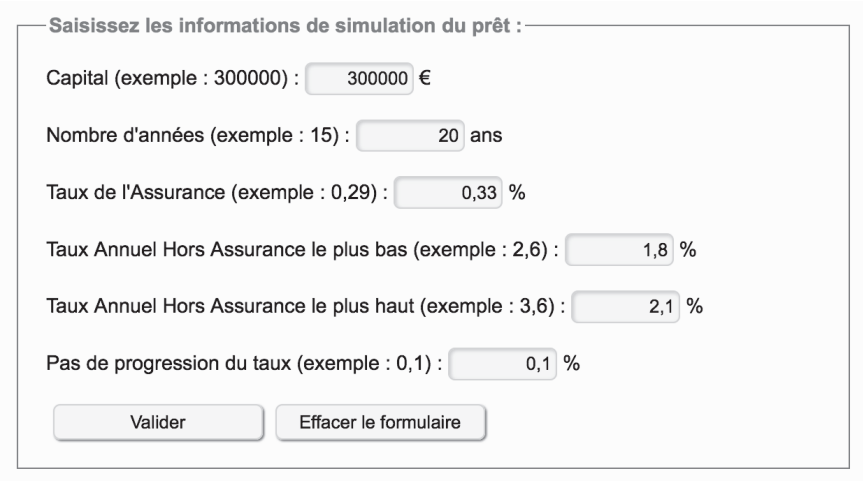


Figure 6.7 – Page de saisie d'un crédit.

Résultat de la simulation

Taux Simulation %	Mensualité Assurance Comprise €	Assurance €	Coût du Crédit en € et en % du capital emprunté	
1,80 %	1 571,90 €	82,50 €	57 456,00 €	19,15 %
1,90 %	1 585,98 €	82,50 €	60 835,20 €	20,28 %
2,00 %	1 600,15 €	82,50 €	64 236,00 €	21,41 %
2,10 %	1 614,40 €	82,50 €	67 656,00 €	22,55 %
Taux Simulation %	Mensualité Assurance Comprise €	Assurance €	Coût du Crédit en € et en % du capital emprunté	

Figure 6.8 – Résultat de la simulation.

Solutions

6.1 Le fichier `switch_note_mention_reel_shell.php` lit trois notes et trois coefficients réels et affiche la moyenne pondérée puis la mention *via* un `switch` sur un sélecteur booléen.

Listing 6.46 – Programme switch_note_mention_reel_shell.php

```
<?php
  echo "Entrez 3 notes        : ";
  fscanf(STDIN,"%f %f %f",$n1,$n2,$n3);
  echo "Entrez 3 coefficients : ";
  fscanf(STDIN,"%f %f %f",$c1,$c2,$c3);
  $Moyenne=( ($n1*$c1)+($n2*$c2)+($n3*$c3) )/($c1+$c2+$c3) ;
  printf("Votre moyenne est : %+6.2f\n",$Moyenne);
  switch (true)
  {
    case (($Moyenne >= 0) && ($Moyenne < 10))  :
              echo "Recalé".PHP_EOL;
              break;
    case (($Moyenne >= 10) && ($Moyenne < 12)) :
              echo "Mention PASSABLE".PHP_EOL;
              break ;
    case (($Moyenne >= 12) && ($Moyenne < 14)) :
              echo "Mention ASSEZ BIEN".PHP_EOL;
              break ;
    case (($Moyenne >= 14) && ($Moyenne < 16)) :
              echo "Mention BIEN".PHP_EOL;
              break ;
    case (($Moyenne >= 16) && ($Moyenne <= 20)):
              echo "Mention TRES BIEN".PHP_EOL;
              break ;
    default: echo "Notes ou Coefficients erronnés".PHP_EOL;
```

```
            break;
  }
?>
```

6.2 `switch_note_mention_reel_web.html` est le formulaire de saisie.

Listing 6.47 – Fichier switch_note_mention_reel_web.html

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Calcul de la moyenne</title>
 </head>
 <body>
  <form action="switch_note_mention_reel_web.php" method="post">
  Entrez 3 notes :<br>
  Premi&egrave;re note  <input type="text" name="n1" size="10" /><br>
  Deuxi&egrave;me note  <input type="text" name="n2" size="10" /><br>
  Troisi&egrave;me note  <input type="text" name="n3" size="10" /><br>
  <br>
  Entrez 3 coefficients :<br>
  Premier  coefficient  <input type="text" name="c1" size="10" /><br>
  Deuxi&egrave;me coefficient  <input type="text" name="c2" size="10" /><br>
  Troisi&egrave;me coefficient  <input type="text" name="c3" size="10" /><br>
  <input type="submit" value="Valider" />
  <input type="reset" value="Effacer le formulaire" />
  </form>
 </body>
</html>
```

La validation exécute le programme `switch_note_mention_reel_web.php` sur les données transmises en méthode POST.

Listing 6.48 – Programme switch_note_mention_reel_web.php

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
   <meta charset="utf-8" />
   <title>Calcul de la moyenne</title>
 </head>
 <body>
 <?php
 define("WEB_EOL","<br>");
 // --- On récupère les données ---
 $n1=$_POST['n1'] ;
 $n2=$_POST['n2'] ;
 $n3=$_POST['n3'] ;
 $c1=$_POST['c1'] ;
 $c2=$_POST['c2'] ;
```

```
$c3=$_POST['c3'] ;
// --- On traite les données ---
$n1=floatval($n1);
$n2=floatval($n2);
$n3=floatval($n3);
$c1=floatval($c1);
$c2=floatval($c2);
$c3=floatval($c3);
// --- On calcule ---
$Moyenne=(($n1*$c1)+($n2*$c2)+($n3*$c3))/($c1+$c2+$c3) ;
printf("Votre moyenne est : %+6.2f",$Moyenne);
echo WEB_EOL;
switch (true)
{
   case (($Moyenne >= 0) && ($Moyenne < 10))  :
            echo "Recalé".WEB_EOL;
            break;
   case (($Moyenne >= 10) && ($Moyenne < 12)) :
            echo "Mention PASSABLE".WEB_EOL;
            break ;
   case (($Moyenne >= 12) && ($Moyenne < 14)) :
            echo "Mention ASSEZ BIEN".WEB_EOL;
            break ;
   case (($Moyenne >= 14) && ($Moyenne < 16)) :
            echo "Mention BIEN".WEB_EOL;
            break ;
   case (($Moyenne >= 16) && ($Moyenne <= 20)):
            echo "Mention TRES BIEN".WEB_EOL;
            break ;
   default: echo "Notes ou Coefficients erronnés".WEB_EOL;
            break;
}
?>
</body>
</html>
```

6.3 Le programme `for_simulateur_credit_shell.php` simule les Mensualités d'un crédit, pour les Taux d'intérêts compris entre un *taux de départ*, et un *taux final* en fonction d'un pas de progression. Les valeurs saisies sont le *Capital emprunté*, le *Nombre d'années*, le *Taux d'intérêt annuel*, le *Taux de départ*, et le *Taux final* de la simulation. Les résultats affichés pour chaque taux sont : la *mensualité*, le *coût mensuel de l'assurance*, le *coût total du crédit* en euros (somme des intérêts mensuels), le *coût total du crédit en %* (rapport du coût total sur le capital emprunté). La boucle `for` utilise un compteur réel.

Listing 6.49 – Programme for_simulateur_credit_shell.php

```
<?php
 // -- Saisie des données initiales --
 echo "Capital: "                    ;
```

```
 fscanf(STDIN,"%f",$Capital)        ;
 echo "Nombre d'années : "          ;
 fscanf(STDIN,"%d",$NbAn)           ;
 printf("Taux de l'Assurance (0.29) :") ;
 fscanf(STDIN,"%f",$TauxAssurance)  ;
 echo "Taux Annuel Hors Assurance le plus bas (ex : 2.6) : " ;
 fscanf(STDIN,"%f",$TauxAnnuel_Min) ;
 echo "Taux Annuel Hors Assurance le plus haut(ex : 3.6) : " ;
 fscanf(STDIN,"%f",$TauxAnnuel_Max) ;
 echo "Pas de progression du taux (ex : 0.1) : " ;
 fscanf(STDIN,"%f",$TauxAnnuel_Pas) ;
 // -- On effectue les calculs et on affiche les résultats --
 $NbMois = $NbAn*12 ;
 $CoutAss = round((($Capital*($TauxAssurance/100))/12),2);
 for ($TauxAnnuel=$TauxAnnuel_Min ; $TauxAnnuel<=($TauxAnnuel_
 Max+$TauxAnnuel_Pas) ; $TauxAnnuel+=$TauxAnnuel_Pas)
 {
  $TauxMensuel  = ($TauxAnnuel/100)/12                  ;
  $calcul1      = $Capital*$TauxMensuel                 ;
  $calcul2      = pow((1+$TauxMensuel),$NbMois)         ;
  $calcul3      = $calcul2-1                            ;
  $MensualiteHA = round(($calcul1*($calcul2/$calcul3)),2);
  $MensualiteAC = $MensualiteHA+$CoutAss                ;
  $CoutCredit   = ($MensualiteHA*$NbMois)-$Capital      ;
  $PourcentCout = round(($CoutCredit/$Capital)*100,2)   ;
  // Affichage des résultats
  echo "--- Simulation avec Taux  = $TauxAnnuel ---".PHP_EOL;
  echo "Mensualité Assurance Comprise : ".$MensualiteAC.PHP_EOL;
  echo "Coût de l'Assurance par mois  : ".$CoutAss.PHP_EOL;
  echo "Coût du crédit en €           : ".$CoutCredit.PHP_EOL;
  echo "Coût du crédit en %           : ".$PourcentCout.PHP_EOL;
 }
?>
```

6.4 Le fichier `for_simulateur_credit_web_style.html` est le formulaire de saisie des données du prêt à simuler. La feuille de styles `style_simulateur_2pages.css` est téléchargeable sur le site de l'éditeur.

Listing 6.50 – Fichier for_simulateur_credit_web_style.html

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
  <meta charset="utf-8" />
  <title>Simulation pr&ecirc;t</title>
  <link href="style_simulateur_2pages.css" rel="stylesheet" type="text/
  css"/>
 </head>
 <body>
  <form action="for_simulateur_credit_web_style.php" method="post">
    <fieldset>
```

```
    <legend>Saisissez les informations de simulation du pr&ecirc;t :</
    legend><br>
      Capital (exemple : 300000) :   <input type="text" name="Capital"
      size="20" /> &euro;<br><br>
      Nombre d'ann&eacute;es (exemple : 15) : <input type="text"
      name="NbAn" size="5" /> ans<br><br>
      Taux de l'Assurance (exemple : 0,29) : <input type="text"
      name="TauxAssurance" size="5" /> %<br><br>
      Taux Annuel Hors Assurance le plus bas  (exemple : 2,6): <input
      type="text" name="TauxAnnuel_Min" size="5" /> %<br><br>
      Taux Annuel Hors Assurance le plus haut (exemple : 3,6): <input
      type="text" name="TauxAnnuel_Max" size="5" /> %<br><br>
      Pas de progression du taux (exemple : 0,1) : <input type="text"
      name="TauxAnnuel_Pas" size="5" /> %<br><br>
      <input type="submit" value="Valider" />
      <input type="reset" value="Effacer le formulaire" />
    </fieldset>
  </form>
 </body>
</html>
```

La validation exécute le programme `for_simulateur_credit_web_style.php` sur les données « texte » transmises en méthode POST. L'absence des données affiche une page d'erreur. Ce contrôle pourrait également être effectué au niveau du formulaire *via* les syntaxes HTML5. La saisie des valeurs réelles peut se noter avec la virgule décimale (notation française), la chaîne de caractères reçue est traitée pour remplacer les « , » par des « . » *via* `str_replace()` puis convertie en réel avec `floatval()`. Les chaînes contenant des entiers sont également converties explicitement en entier *via* `intval()`.

Listing 6.51 – Programme for_simulateur_credit_web_style.php

```
<!DOCTYPE html>
<html>
 <head> <!-- Entête HTML -->
  <meta charset="utf-8" />
  <title>Simulation pr&ecirc;t</title>
  <link href="style_simulateur_2pages.css" rel="stylesheet" type="text/
  css"/>
 </head>
 <body>
 <?php
  // --- On récupère les données ---
  $Capital     = $_POST['Capital']        ;
  $NbAn        = $_POST['NbAn']           ;
  $TAssurance  = $_POST['TauxAssurance']  ;
  $TAnnuel_Min = $_POST['TauxAnnuel_Min'] ;
  $TAnnuel_Max = $_POST['TauxAnnuel_Max'] ;
  $TAnnuel_Pas = $_POST['TauxAnnuel_Pas'] ;
  if(empty($Capital) || empty($NbAn) || empty($TAnnuel_Min) ||
```

```
empty($TAnnuel_Max) || empty($TAnnuel_Pas) || empty($TAssurance))
{
  ?>
  <fieldset>
  <legend>Valeurs &agrave; renseigner :</legend><br/>
  <?php
  if(empty($Capital))   echo "Capital <br/>";
  if(empty($NbAn))      echo "Nombre d'ann&eacute;es <br/>";
  if(empty($TAnnuel_Min)) echo "Taux Annuel Hors Assurance initial
  <br/>";
  if(empty($TAnnuel_Max)) echo "Taux Annuel Hors Assurance final
  <br/>";
  if(empty($TAnnuel_Pas)) echo "Intervalle de progression du taux
  <br/>";
  if(empty($TAssurance))  echo "Taux de l'Assurance <br/>";
  ?>
  </fieldset>
<?php
}
else
{
  ?>
  <table summary="SOMMAIRE : Résultat de la simulation">
  <caption>R&eacute;sultat de la simulation</caption>
  <thead>
    <tr>
      <th>Taux Simulation %</th>
      <th>Mensualit&eacute; Assurance Comprise &euro;</th>
      <th>Assurance &euro;</th>
      <th colspan="2">Co&ucirc;t du Cr&eacute;dit en &euro; et en %
      du capital emprunt&eacute;</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <th>Taux Simulation %</th>
      <th>Mensualit&eacute; Assurance Comprise &euro;</th>
      <th>Assurance &euro;</th>
      <th colspan="2">Co&ucirc;t du Cr&eacute;dit en &euro; et en %
      du capital emprunt&eacute;</th>
    </tr>
  </tfoot>
  <?php
  // --- On traite les données ---
  // --- Remplacement de la virgule par un point décimal ---
  $Capital     = str_replace(",",".",$Capital)    ;
  $TAnnuel_Min = str_replace(",",".",$TAnnuel_Min);
  $TAnnuel_Max = str_replace(",",".",$TAnnuel_Max);
  $TAnnuel_Pas = str_replace(",",".",$TAnnuel_Pas);
  $TAssurance  = str_replace(",",".",$TAssurance) ;
```

```
    // --- Conversion dans le bon type de données ---
    $Capital     = floatval($Capital)     ;
    $NbAn        = intval($NbAn)          ;
    $TAnnuel_Min = floatval($TAnnuel_Min) ;
    $TAnnuel_Max = floatval($TAnnuel_Max) ;
    $TAnnuel_Min = floatval($TAnnuel_Min) ;
    $TAnnuel_Pas = floatval($TAnnuel_Pas) ;
    $TAssurance  = floatval($TAssurance)  ;
    // -- On effectue les calculs et on affiche les résultats
    $NbMois = $NbAn*12 ;
    $CoutAss = round((($Capital*($TAssurance/100))/12),2);
    for ($TAnnuel=$TAnnuel_Min ; $TAnnuel<=$TAnnuel_Max ;
    $TAnnuel+=$TAnnuel_Pas)
    {
     $TauxMensuel  = ($TAnnuel/100)/12                    ;
     $calcul1      = $Capital*$TauxMensuel                ;
     $calcul2      = pow((1+$TauxMensuel),$NbMois)        ;
     $calcul3      = $calcul2-1                           ;
     $MensualiteHA = round(($calcul1*($calcul2/$calcul3)),2);
     $MensualiteAC = $MensualiteHA+$CoutAss               ;
     $CoutCredit   = ($MensualiteHA*$NbMois)-$Capital     ;
     $PourcentCout = ($CoutCredit/$Capital)*100           ;
     // on prépare les résultats au format français
     $TauxA=number_format($TAnnuel,2,","," ")." %"          ;
     $MensHA=number_format($MensualiteHA,2,","," ")." &euro;";
     $MensAC=number_format($MensualiteAC,2,","," ")." &euro;";
     $CoutA=number_format($CoutAss,2,","," ")." &euro;"     ;
     $CoutC=number_format($CoutCredit,2,","," ")." &euro;"  ;
     $PcentCout=number_format($PourcentCout,2,","," ")." %" ;
     ?>
     <tr>
       <td><?php echo $TauxA    ; ?></td>
       <td><?php echo $MensAC   ; ?></td>
       <td><?php echo $CoutA    ; ?></td>
       <td><?php echo $CoutC    ; ?></td>
       <td><?php echo $PcentCout; ?></td>
     </tr>
     <?php
    }
    ?>
    </table>
    <?php
  }
 ?>
 </body>
</html>
```