Algorithmique avec le langage de programmation Php

Algorithme

Description d'un processus de calcul :

- fini
- chaque étape est définie complètement et sans ambiguïté
- qui résout un problème clairement spécifié

Algorithmique avec le langage de programmation Php

Algorithme

Description d'un processus de calcul :

- fini
- chaque étape est définie complètement et sans ambiguïté
- qui résout un problème clairement spécifié

Programme

Mise en œuvre d'un algorithme avec un langage de programmation. Le texte d'un programme (aspect statique) décrit une suite de changement d'états (aspect dynamique). Les constituants d'un programme sont :

- des objets informatiques : variables, constantes, ...
- des actions : instructions, structures de contrôle, ...

Le langage PHP

Documentation de référence

http ://www.php.net/manual/fr/index.php

Historique de PHP

- 1995 : Diffusion grand public de PHP/FI (Personnal Home Page/Forms Interpreter) une bibliothèque de script PERL créée par Rasmus Lerdof
- 1997 : PHP/FI 2.0 est une nouvelle version réécrite en C
- 1998 : PHP 3.0 (PHP : Hypertext Preprocessor) est à nouveau réécrit par Andi Gutmans, Zeev Suraski et Rasmus Lerdof afin d'être efficace et extensible
- 2000 : PHP 4.0
- 2004 : PHP 5.0 permet la programmation par objets

Caractéristique du langage PHP

- Interprété ou compilé juste à temps
- Dynamiquement typé
- Forte intégration au Web puisque le code peut être intégré à dans un page HTML

Généralités

- Tout code PHP doit être inclus dans une balise <?php ... ?>
- Les commentaires multi-lignes sont entre les signes /* ... */
- Les commentaires sur une seule ligne commencent par // ou #

Variables

```
$mavariable = <expression>;
$foo = "0";  // $foo est une chaîne de caractère
$foo = 3;    // $foo est un entier
$foo = 3.14116; // $foo est un réel
```

- Un nom de variable commence toujours par un $
- Attention à la casse ($i différent de $I)
- Une variable n'a pas besoin d'être déclarée

Constantes

```
define("PI", 3.141166); // definition de constante PI
$foo=3*PI; // utilisation de la constante PI
```

- Les constantes sont définies et accessibles à tout endroit du code, globalement
- Les constantes ne peuvent pas être redéfinies ou indéfinies une fois qu'elles ont été définies

Types numériques

```
$i = 1;  // entier en notation décimale
$i = 3.14;  // flottant
$i = 0.3e-3; // Notation exponentielle (0,0003)

$a = 3; $b = 5 ;
$c = $a + $b * 5;

$c = 4;
$d = ++$c; // d vaut 5
$d = $c--; // d vaut 4
```

- Les opérateurs arithmétiques sont : + (addition), - (soustraction), * (multiplication), / (division), % (modulo), ++ (incrémentation), −− (décrémentation)
- La précédence des opérateurs est disponible dans la documentation PHP. Dans le doute mettez des parenthèses !

Chaînes de caractères

```
$c = 'Une chaine immuable sans caractère d\'echapement
et acceptant les sauts de lignes';

$i = 3;
// une chaine dans laquelle on peut intégrer des variables
$c = "$i petits \n cochons";

// concatener des chaines
$a = "Hello ";
$b = $a . "World"; // maintenant $b contient "Hello World!"
$b .= "!"; // opérateur d'assignation

// afficher des chaines à l'écran
echo $b;
```

Booléens

```
$b = TRUE;
$b = True;
$c = !$b;
```

- La valeur 0 est considérée comme fausse et toutes les autres valeurs comme vraies
- TRUE, FALSE sont insensibles à la casse
- Les opérateurs logiques sont : ! (négation), || or (ou), && and (et), xor (ou exclusif)

Types et valeurs

Attention aux types

- Même si l'on ne déclare pas le type d'une variable, elle en a un !
- Les types scalaires sont : bool, entier, double et string
- On peut tester le type d'une variable grâce à des fonctions prédéfinies : gettype(var), is_double(var), is_integer(var), is_string(var), ...

La valeur NULL

- Représente l'absence de valeur
- Peut être affectée à une variable
- Est la valeur d'une variable non affectée

Conversions automatiques

```
$foo = "0";  // $foo est une chaîne de caractère
$foo++;      // $foo est une chaîne de caractère ("1")
$foo += 1;   // $foo est un entier (2)
$foo = $foo + 1.3;  // $foo est un réel (3.3)
$foo = 5 + "10 Little Piggies"; // $foo est un entier (15)
```

Attention aux types

- Même si l'on ne déclare pas le type d'une variable, elle en a un !
- Attention aux conversions automatiques !

Conversions manuelles

```
$foo = "10";  // $foo est une chaîne de caractère
$foo = (int) $foo; // $foo est un entier (10)
$foo = (float) $foo; // $foo est un flottant
```

Un premier script : Hello World

```
<?php
    error_reporting(E_ALL); # signale toutes les erreurs

    $message = 'Hello';
    $message .= " World !";
    echo $message;
?>
```

Expression vs Instruction

Définition

Une expression est une construction du langage qui produit une valeur. Les variables sont des expressions.

```
$foo = "10";  // est une expression qui vaut 10
$bar = $foo = 10;
$bar;
3*4;
```

Test

```
if ( <expression> )
{
// Bloc si expression vraie
}
else
{
// Bloc si expression fausse
}
```

- Opérateurs de comparaison sont == (égalité), ! = (different), <, >, <=, >=
- Un bloc est une séquence d'instructions

Égalité vs Affectation

```
$i = 3; $j = 4;
if ( $i = $j )
{
echo "$i et $j sont égaux";
}
else
{
echo "$i et $j ne sont pas égaux";
}
```

Égalité vs Affectation

```
$i = 3; $j = 4;
if ( $i = $j )
{
echo "$i et $j sont égaux";
}
else
{
echo "$i et $j ne sont pas égaux";
}
```

- Une affectation (=) a une valeur (4 dans l'exemple précédent)
- Ne pas confondre avec le test d'égalité (==)

Exercice

Énoncé

Un triplet d'entiers (a, b, c) est pythagoricien si l'équation de Pythagore $a^2 + b^2 = c^2$ est vérifiée. Écrire un programme (en PHP) qui vérifie que trois entiers a, b, c sont pythagoriciens. Le résultat sera stocké dans une variable booléenne.

Exercice

Énoncé

Un triplet d'entiers (a, b, c) est pythagoricien si l'équation de Pythagore $a^2 + b^2 = c^2$ est vérifiée. Écrire un programme (en PHP) qui vérifie que trois entiers a, b, c sont pythagoriciens. Le résultat sera stocké dans une variable booléenne.

Corrigé

```
$a = 3; $b = 4; $c = 5;
$pythagoricien = (($a*$a)+($b*$b))==($c*$c) );
echo "$a, $b, $c est-il pythagoricien ? $pythagoricien";
```

Opérateur ternaire

```
$i = 3 ; $j = 4;
echo ($i<=$1$j ? "i" : "j")." est le plus petit";
```

- Permet d'utiliser une structure conditonnelle comme une expression

Instruction elseif

```
if ($a > $b) {
   echo "a est plus grand que b";
} elseif ($a == $b) {
   echo "a est égal à b";
} else {
   echo "a est plus petit que b";
}
```

Exercice

Les deux programmes suivants sont-ils équivalents ?

```
if (b1) { i1; }
elsif (b2) {
i2;
}
i3;
```

```
if ( b1 ) { i1; }
if ( b2 ) { i2; }
i3;
```

Exercice

Les deux programmes suivants sont-ils équivalents ?

```
if (b1) { i1; }
elsif (b2) {
i2;
}
i3;
```

```
if ( b1 ) { i1; }
if ( b2 ) { i2; }
i3;
```

Réponse

NON ! Par exemple :

- b1 : \$a > \$b
- i1 : \$a = \$b
- b2 : \$a <= \$b
- i2 : \$a = 0

Instruction switch

```
switch (<expression>) {
case <valeur1>:
    // bloc d'instructions 1
    break;
case <valeur2>:
    // bloc d'instructions 2
    break;
case <valeur3>:
  // bloc d'instructions 3
break;
default:
// bloc d'instructions par défaut
break;
}
```

Instruction for

Répeter un ensemble d'instruction un nombre de fois calculable.

```
for ( $x=0 ; $x<10 ; $x++ )
{
// bloc d'instructions
}
```

Le for est constitué de trois parties séparées par des points virgules :

- La partie initialisation ($x=0)
- La partie test de fin ($x¡10)
- La partie action ($x++)

Instruction for

Mieux vaut éviter de condenser même si cela est possible

```
for ( $a=1, $b=6 ; $a<$b ; $a++, print("a = $a") ) ;
```

Exercice

Écrire un programme qui à partir d'une température exprimée en degrés Celsius C, la transforme en degrés Farnheit F sachant que : F = C*1.8+32.
Écrire ensuite un programme qui réalise la transformation inverse.

Exercice

Écrire un programme, qui étant donné un temps exprimé en secondes, le convertit en heures, minutes et secondes et l'affiche sous le format : ___H ___MN ___SEC.

Instruction While

Une boucle while s'utilise lorsqu'on ne connaît pas le nombre d'itérations à effectuer.

```
while (<expression>) {
    <instructions>
}
```

```
$i = 1;
while ($i <= 10) {
   echo $i++;
   /* La valeur affichée est $i avant l'incrémentation */
}
```

Exercice

- Saisir 10 entiers et afficher leur somme

Exercice

- Saisir 10 entiers et afficher leur somme
- Saisir N entiers et afficher leur somme (la valuer de N étant saisie préalablement)

Exercice

- Saisir 10 entiers et afficher leur somme
- Saisir N entiers et afficher leur somme (la valuer de N étant saisie préalablement)
- Saisir une succession d'entiers (la fin des entrées est marquée par la frappe d'un 0) et afficher leur somme

Exercice

- Saisir 10 entiers et afficher leur somme
- Saisir N entiers et afficher leur somme (la valuer de N étant saisie préalablement)
- Saisir une succession d'entiers (la fin des entrées est marquée par la frappe d'un 0) et afficher leur somme
- Même chose que précédemment mais afficher aussi leur moyenne, le minimum et le maximum des entiers saisis

Instruction Do-While

Une boucle est utilisée lorsqu'on veut que les instruction soient exécutées au moins une fois.

```
do {
    <instructions>
} while( <expression> );
```

```
$i = 0;
do {
   echo $i;
} while ($i > 0);
```

Définition d'une fonction

```
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
   // instructions
}

// appel d'une fonction
foo($param_reel_1, /* ..., */ $param_reel_n);
```

- A chaque appel d'une fonction, son corps est exécuté en substituant ses paramètres formels aux paramètres réels
- Les arguments d'une fonction sont passés par copie (mécanisme par défaut)
- Si une fonction contient une instruction *return* $<$ *expression* $>$ alors elle retourne la valeur de cette expression

Exemple complet

pair.php

```
<?php
error_reporting(E_ALL);
include('io.php');

function is_pair( $nb ) {
        return ($nb % 2) == 0;
    }

    echo (is_pair(getArgv(1))? "pair" : "impair") ."\n";
?>
```

Exemple complet

pair.php

```
<?php
error_reporting(E_ALL);
include('io.php');

function is_pair( $nb ) {
        return ($nb % 2) == 0;
    }

    echo (is_pair(getArgv(1))? "pair" : "impair") ."\n";
?>
```

Execution du programme

```
login@machine $ php pair.php 4
pair
login@machine $
```

Quelques exemples de fonctions

```
$a = 3;

// affiche le type d'une expression
// ainsi que sa valeur
var_dump($a);

// retourne le résultat de 2 à la puissance 3
// ou FAUX si le calcul est impossible
$res = pow($arg1,$arg2);
```

Le passage par copie

Attention, une fonction ne modifie pas ses paramètres !

Exemple

```
function swap( $a, $b ) {
        $c = $b; $b = $a ; $a = $c;
}

$i = 1; $j = 2;
swap($i,$j);
echo "$i $j";
```

Les paramètres par défauts

Il est possible de spécifier des valeurs par défaut pour les derniers paramètres d'une fonction lors de sa définition.

Exemple

```
function afficher( $texte, $saut = 1 ) {
        echo $texte;
    for( $i = 0 ; $i < $saut ; $i++)
        echo "\n";
}

afficher("Hello", 0);
afficher(" World !");
```

Portée des variables globales et locales

Une variable globale n'est pas visible dans une fonction, pour qu'elle le soit il faut le préciser avec le mot clé *global*.

sport.php

```
// fonction Sport1 $sport est une variable locale
function sport1(){
  $sport = "volley";
  echo "aimez-vous le $sport ?\n";
}

// fonction Sport2 $sport est une variable globale
function sport2(){
  global $sport;
  echo "aimez-vous le $sport ?\n";
}

$sport  = "badmington";
sport1();
sport2();
```

Exercice

Soit la suite définie par :

$$u_0 = 1, u_1 = 1, u_{n+2} = u_{n+1} + u_n \text{ (suite de fibonacci)}$$

Écrivez une fonction qui calcule la valeur de rang k de cette suite

Tableaux à une dimension

Un tableau en PHP est en fait une association ordonnée. Une association est un type qui fait correspondre des valeurs à des clés.

```
$tab1 = array("A","B","C");

$tab2[0] = "A";
$tab2[1] = "B";
$tab2[2] = "C";

$tab3[] = "A";
$tab3[] = "B";
$tab3[] = "C";
```

0	"A"
1	"B"
2	"C"

Remarques

- les indices commencent à 0
- tab1, tab2 et tab3 ont tous la même structure
- ces 3 syntaxes peuvent être combinées

Quelques fonctions utiles

```php
$tab = array(20,12,4,17,-3);

// affiche récursivement le tableau
echo "tab : "; print_r($tab);

// renvoie le nombre d'éléments dans le tab
echo "nb elt = " . count($tab) . "\n";

// enlève la valeur associée à la clef 0
unset($tab[0]);
echo "tab : "; print_r($tab);

// créé un nouveau tableau à partir d'un autre
$tab2 = array_values($tab);
echo "tab2 : "; print_r($tab2);

// découpe une chaîne en tableau
$tab = explode("/", "12/06/2007");
print_r($tab);
echo gettype($tab[0]) . "\n";
```

```
tab : Array
(
    [0] => 20
    [1] => 12
    [2] => 4
    [3] => 17
    [4] => -3
)
nb elt = 5
tab : Array
(
    [1] => 12
    [2] => 4
    [3] => 17
    [4] => -3
)
tab2 : Array
(
    [0] => 12
    [1] => 4
    [2] => 17
    [3] => -3
)
Array
(
    [0] => 12
    [1] => 06
    [2] => 2007
)
string
```

Parcours de tableaux avec l'instruction for

Exemple de parcours total

```
$tab = array(1,2,3,4,5,6);

// fonction mélangeant le tableau
shuffle($tab);

echo "valeurs dans tab : ";
for( $i = 0 ; $i<count($tab) ; $i++){
   echo $tab[$i];
}
echo "\n";

// les fonctions max et min existent
echo "nb min = ".max($tab)."\n";
echo "nb max = ".min($tab)."\n";
```

Parcours de tableaux avec l'instruction while

Exemple de parcours partiel

```
$tab = array(1,2,3,4,5,6);
$i = 0;
echo "3 premieres valeurs dans tab : ";
while( $i<count($tab) and ($i<3)  ){
   echo $tab[$i] . " ";
   $i++;
}
echo "\n";
```

Parcours de tableaux avec l'instruction foreach

L'instruction *foreach* simplifie l'écriture des parcours totaux de tableaux.

Exemple

```
$tab = array(5,2,5,7,4,6);

foreach($tab as $v){
   echo $v. " ";
}
echo "\n";
```

Les tableaux associatifs

Les «clés» d'un tableau peuvent être soit des entiers, soit des chaînes de caractères.

```
$personne = array(
    'prenom' => 'john',
    'nom' => 'doe',
    'age' => 20
);

echo "Bonjour " .
    $personne['prenom'] . "\n";

// nouvelle construction syntaxique
echo "Bonjour M. {$personne['nom']}\n";
```

'prenom'	'john'
'nom'	'doe'
'age'	20

Remarque

Par défaut, la clef est un entier

Les tableaux avec des clés explicites et implicites

```
$t = array(5 => 43, 32, 56, "b" => 12);
$u = array(5 => 43, 6 => 32, 7 => 56, "b" => 12);

// $t et $u sont identiques

echo $u["b"];
```

Les tableaux à 2 dimensions

```
$matrice = array(
array( 1, 18, 6 ),
array( -1, 1, 8 ),
array( 13, 18, 3 )
);
echo $matrice[1][2];

$personnes = array(
    array( 'prenom' => 'john', 'nom' => 'doe', 'age' => 20 ),
    array( 'prenom' => 'luke', 'nom' => 'skywalker', 'age' => 17 ),
);
echo $personnes[0]['prenom'];
```

Création d'une liste de personnes

```
$ php personnes.php
Saisir un prénom : john
Saisir un nom : doe
Saisir une autre personne (o|n) : o
Saisir un prénom : luke
Saisir un nom : skywalker
Saisir une autre personne (o|n) : n
Array
(
    [0] => Array
        (
            [prenom] => john
            [nom] => doe
        )

    [1] => Array
        (
            [prenom] => luke
            [nom] => skywalker
        )

)
```

Créer un tableau de tableau associatif

personnes.php

```
function saisie_personne() {
    echo 'Saisir un prénom : '; $prenom = getLine();
    echo 'Saisir un nom : '; $nom = getLine();

    return array( 'prenom' => $prenom , 'nom' => $nom );
}

function saisie_personnes() {
    $continue = 'o';
    $personnes = array();

    while($continue == 'o') {
        $personnes[] = saisie_personne();

        echo 'Saisir une autre personne (o|n) : '; $continue = getLine();
    }
    return $personnes;
}

$personnes = saisie_personnes();
print_r($personnes);
```

Parcours simplifié d'un tableau associatif

personnes.php

```
function saisie_personne() { ... }
function saisie_personnes() { ... }
funtion affiche_personne($personne) {
    foreach ($personne as $k => $v)
        echo "{$k} => {$v}\n";
}

$personnes = saisie_personnes();
foreach( $personnes as $personne )
    affiche_personne($personne);
```

Exécution

```
$ php liste.php
Saisir un prénom : john
Saisir un nom : doe
Saisir une autre personne (o|n) : o
...
prenom => john
nom => doe
prenom => luke
nom => skywalker
```

La fonction list

```
$info = array('café', 'noir', 'cafeine');

list($boisson, $couleur, $composant) = $info;

echo "Je bois du $boisson $couleur avec de la $composant\n";
```

Exemple d'utilisation de la fonction list

```
$personnes = ...; // saisie une liste de personnes

foreach $personnes as $personne {
    while (list ($prenom, $nom, $age) = $personne) {
        echo "$nom $prenom $age\n";
    }
}
```