



Ressources **informatiques**

Algorithmique

Techniques fondamentales
de programmation

Exemples en **PHP**

(nombreux exercices corrigés)

2^{ème} édition

BTS,
DUT Informatique

Olivier ROLLET
Sébastien ROHAUT

Téléchargement
www.editions-eni.fr



Les éléments à télécharger sont disponibles à l'adresse suivante :
<http://www.editions-eni.fr>
Saisissez la référence ENI de l'ouvrage **RI2PALG** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

Chapitre 1 Introduction à l'algorithmique

1. Les fondements de l'informatique	13
1.1 Architecture de Von Neumann	13
1.2 La machine de Turing	17
1.3 Représentation interne des instructions et des données	19
1.3.1 Le binaire	19
1.3.2 Les octets et les mots	22
1.3.3 L'hexadécimal	23
2. L'algorithmique	24
2.1 Programmer, c'est un art	24
2.2 Définition : l'algorithme est une recette	26
2.3 Pourquoi utiliser un algorithme ?	27
2.4 Le formalisme	28
2.4.1 La représentation graphique	29
2.4.2 L'algorithme sous forme de texte	30
2.5 La complexité	32
3. Les langages d'implémentation	35
3.1 Quel langage ?	35
3.2 Classifications des langages	37
3.2.1 Haut niveau, bas niveau	38
3.2.2 Diverses classifications	39
3.2.3 Compilé ou interprété	39

2 **Algorithmique**

(avec des exemples en PHP)

3.3	La machine virtuelle	41
3.4	PHP	42
3.4.1	Les avantages	42
3.4.2	Historique	42
3.4.3	Informations pratiques	43
3.4.4	Pages dynamiques.	44
3.4.5	Installer le nécessaire	44
3.4.6	Un premier programme PHP	46
4.	Exercices	47

Chapitre 2

Les variables et opérateurs

1.	Les variables	49
1.1	Principe	49
1.2	Déclaration	52
1.3	Types.	52
1.3.1	Les nombres	53
1.3.2	Autres types numériques.	56
1.3.3	Les caractères	57
1.3.4	Le type booléen.	59
1.4	Affectation	60
1.4.1	Affectation de valeurs	60
1.4.2	Affectation de variables	63
1.5	Saisie et affichage	64
1.6	Les constantes.	67
2.	Opérateurs et calculs	68
2.1	Les affectations	68
2.2	Les opérateurs arithmétiques.	68
2.3	Les opérateurs booléens	74
2.4	Les opérateurs de comparaison	77
2.4.1	L'égalité	77
2.4.2	La différence	79

2.4.3	Inférieur, supérieur	79
2.5	Le cas des chaînes de caractères	80
3.	Pour aller plus loin	81
3.1	Les nombres négatifs	81
3.2	La représentation des nombres réels	83
3.3	Les dates	87
3.4	Les caractères	88
4.	Types et langages	90
4.1	Langages typés ou non	90
4.2	La gestion de la mémoire	91
5.	Exercices	92

Chapitre 3
Tests et logique booléenne

1.	Les tests et conditions	95
1.1	Principe	95
1.2	Que tester ?	97
1.3	Tests SI	98
1.3.1	Forme simple	98
1.3.2	Forme complexe	100
1.4	Tests imbriqués	102
1.5	Choix multiples	106
1.6	Des exemples complets	109
1.6.1	Le lendemain d'une date	109
1.6.2	La validité d'une date	113
1.6.3	L'heure dans n secondes	114
2.	L'algèbre booléen	119
2.1	L'origine des tests	119
2.2	Petites erreurs, grosses conséquences	121
2.2.1	Ariane 5	121
2.2.2	Mars Climate Orbiter	122

4 **Algorithmique**

(avec des exemples en PHP)

2.3	George Boole	122
2.4	L'algèbre	123
2.4.1	Établir une communication	123
2.4.2	La vérité	125
2.4.3	La loi ET	125
2.4.4	La loi OU	126
2.4.5	Le contraire	127
2.4.6	Les propriétés	127
2.4.7	Quelques fonctions logiques	131
2.4.8	Avec plus de deux variables	134
2.5	Une dernière précision	137
3.	Exercices	139

Chapitre 4 **Les boucles**

1.	Les structures itératives	141
1.1	Définition	141
1.2	Quelques usages simples	142
2.	Tant Que	143
2.1	Structure générale	143
2.2	Boucles infinies et break	145
2.3	Des exemples	147
2.3.1	Une table de multiplication	147
2.3.2	Une factorielle	148
2.3.3	x à la puissance y	149
2.3.4	Toutes les tables de multiplication	152
2.3.5	Saisie de notes et calcul de moyennes	153
2.3.6	Rendez la monnaie	159
2.3.7	Trois boucles	163

- 3. Répéter ... Jusqu'à 164
 - 3.1 Différences fondamentales..... 164
 - 3.2 Quelques exemples adaptés 166
 - 3.2.1 La factorielle 166
 - 3.2.2 Les trois boucles 167
- 4. Pour ... Fin Pour 168
 - 4.1 Une structure pour compter... 168
 - 4.2 ... mais pas indispensable 169
 - 4.3 Quelle structure choisir ? 169
 - 4.4 Un piège à éviter 170
 - 4.5 Quelques exemples 171
 - 4.5.1 De nouveau trois boucles..... 171
 - 4.5.2 La factorielle 172
 - 4.5.3 Racine carrée avec précision..... 173
 - 4.5.4 Calcul du nombre PI..... 176
- 5. Exercices 178

Chapitre 5
Les tableaux et structures

- 1. Présentation 181
 - 1.1 Principe et définitions 181
 - 1.1.1 Simplifier les variables 181
 - 1.1.2 Les dimensions 183
 - 1.1.3 Les types 184
 - 1.1.4 Déclaration 184
 - 1.1.5 Utilisation 185
 - 1.1.6 Les tableaux dynamiques..... 186
 - 1.2 PHP et les tableaux..... 187
 - 1.3 Représentation en mémoire..... 191
 - 1.3.1 Représentation linéaire..... 191
 - 1.3.2 Représentation par référence 193

6 **Algorithmique**

(avec des exemples en PHP)

2.	Manipulations simples	195
2.1	Recherche d'un élément	195
2.2	Le plus grand/petit, moyenne	197
2.3	Le morpion	198
3.	Algorithmes avancés.	203
3.1	Les algorithmes des tris	203
3.1.1	Principe	203
3.1.2	Le tri par création	204
3.1.3	Le tri par sélection	204
3.1.4	Le tri à bulles	206
3.1.5	Le tri par insertion	210
3.1.6	Le tri Shell	213
3.1.7	Le tri Batcher.	215
3.2	Recherche par dichotomie	216
4.	Structures et enregistrements	219
4.1	Principe	219
4.2	Déclaration	220
4.2.1	Type structuré	220
4.2.2	Enregistrement	221
4.3	Utiliser les enregistrements	222
4.3.1	Utiliser les champs	222
4.3.2	Un enregistrement dans une structure	224
4.3.3	Un tableau dans une structure	225
4.4	Les tableaux d'enregistrements	227
4.4.1	Les tables	227
4.4.2	Une table comme champ	228
4.5	Et PHP ?	229
5.	Exercices	231

Chapitre 6 Les sous-programmes

1. Présentation	235
1.1 Principe	235
1.2 Déclaration et définition	237
1.2.1 Dans un algorithme	237
1.2.2 En PHP	238
1.3 Appel	239
1.4 Fonctions et procédures	240
1.4.1 Les procédures	241
1.4.2 Les fonctions	241
1.5 Variables locales et globales	244
1.5.1 Les variables locales	244
1.5.2 Les variables globales	245
1.5.3 Variables globales et PHP	246
1.6 Paramètres	247
1.6.1 Les procédures	247
1.6.2 Les fonctions	250
1.6.3 Paramètres et PHP	252
1.6.4 Petite application fonctionnelle	253
1.7 Sous-programmes prédéfinis	256
1.7.1 Un choix important	256
1.7.2 Quelques exemples	257
1.8 Dernier cas : les tableaux	261
2. Les sous-programmes récursifs	263
2.1 Principe	263
2.2 Un premier exemple : la factorielle	264
2.3 Un exemple pratique : les tours de Hanoi	269
3. Exercices	271

Chapitre 7 **Les fichiers**

1. Les différents fichiers	273
1.1 Préambule	273
1.2 Problématique	274
1.3 Définition	275
1.4 Les formats	275
1.4.1 Types de contenus	275
1.4.2 Le fichier binaire	277
1.4.3 Le fichier texte	278
1.4.4 Quel format utiliser ?	280
1.5 Les accès aux fichiers	281
1.5.1 Séquentiel	281
1.5.2 Accès direct	282
1.5.3 Indexé	282
1.5.4 Autre ?	282
2. Les enregistrements	283
2.1 Les délimiteurs	283
2.2 Largeur fixe	286
2.3 Principes d'accès	287
2.3.1 Étapes de base	287
2.3.2 Identificateurs de fichiers et canaux	288
2.3.3 Les modes d'ouverture	289
3. Fichier texte séquentiel	290
3.1 Ouvrir et fermer un fichier	290
3.2 Lire et écrire des enregistrements	291
3.2.1 Lecture	291
3.2.2 Écriture	293
3.3 Les enregistrements structurés	297
3.4 Exemple en PHP	300
4. Exercices	302

Chapitre 8
Notions avancées

- 1. Les pointeurs et références 303
 - 1.1 Rappels sur la mémoire et les données 303
 - 1.1.1 Structure de la mémoire. 303
 - 1.1.2 PHP : des limites qui n'en sont pas 305
 - 1.1.3 Brefs exemples en C 305
 - 1.2 Le pointeur 306
 - 1.2.1 Principe et définition 306
 - 1.2.2 Le C roi des pointeurs. 308
 - 1.2.3 Applications 309
 - 1.3 Notation algorithmique 312
 - 1.3.1 Déclarer et utiliser les pointeurs 312
 - 1.3.2 Allocation dynamique 314
 - 1.4 PHP et les références. 316
 - 1.4.1 Différences entre le C et PHP 316
 - 1.4.2 Les références 317
 - 1.4.3 Références sur structures 320
 - 1.4.4 Le piège en PHP. 321
 - 1.4.5 La valeur null 322
- 2. Les listes chaînées 324
 - 2.1 Listes chaînées simples 324
 - 2.1.1 Principe 324
 - 2.1.2 Création. 327
 - 2.1.3 Parcours de la liste 329
 - 2.1.4 Recherche 330
 - 2.1.5 Ajout d'un élément. 331
 - 2.1.6 Suppression d'un élément 335
 - 2.1.7 Supprimer toute la liste 338
 - 2.1.8 Parcours récursif 338
 - 2.2 L'implémentation en PHP 339

2.3	Autres exemples de listes	343
2.3.1	Listes circulaires	343
2.3.2	Listes d'éléments triés	343
2.3.3	Listes doublement chaînées	343
2.3.4	Files et piles	344
3.	Les arbres	345
3.1	Principe	345
3.2	Définitions	347
3.2.1	Base	347
3.2.2	Terminologie	347
3.2.3	Description horizontale	348
3.2.4	Description verticale	348
3.2.5	L'arbre binaire	348
3.3	Parcours d'un arbre	349
3.4	Arbre binaire ordonné	352
3.4.1	Principe	352
3.4.2	Recherche d'un élément	352
3.4.3	Ajout d'un élément	354
3.4.4	Suppression d'un nœud	355
3.5	Exemples de tri	356
3.5.1	Tri par fusion	356
3.5.2	Tri rapide	357
4.	Exercices	359

Chapitre 9

Une approche de l'objet

1.	Principe de l'objet, une notion évidente	361
1.1	Avant de continuer	361
1.2	Rappels sur la programmation procédurale	362
1.2.1	Les données	362
1.2.2	Les traitements	363

1.3	L'objet	363
1.3.1	Dans la vie courante	363
1.3.2	En informatique	364
1.4	Classe, objets	367
1.5	Déclaration et accès	369
1.6	Les méthodes	371
1.7	Portée des membres	372
1.8	Encapsulation des données	374
1.9	L'héritage	376
1.9.1	Principe	376
1.9.2	Commerce	378
1.9.3	Hiérarchie	379
1.9.4	Simple ou multiple	380
1.10	Le polymorphisme	381
1.10.1	Principe	381
1.10.2	Le polymorphisme ad hoc	381
1.10.3	Le polymorphisme d'héritage	382
1.10.4	Le polymorphisme paramétrique	384
2.	Manipuler les objets	385
2.1	Les constructeurs	385
2.1.1	Déclaration	385
2.1.2	Appel implicite	386
2.1.3	L'héritage	388
2.2	Les destructeurs	390
2.3	Les attributs statiques	391
2.4	Classes et méthodes abstraites	393
2.5	Interfaces	396
3.	L'objet en PHP	398
3.1	Les langages objet	398
3.2	Déclaration des classes et objets	399
3.3	Héritage	402
3.4	Interfaces	404

4. Exercices	407
--------------------	-----

Chapitre 10

Corrigés des exercices

1. Introduction à l'algorithmique	413
2. Les variables et opérateurs	415
3. Tests et logique booléenne	420
4. Les boucles	427
5. Les tableaux et structures	436
6. Les sous-programmes	442
7. Les fichiers	446
8. Notions avancées	448
9. Une approche de l'objet	451

Index	475
-------------	-----

Chapitre 3

Tests et logique booléenne

1. Les tests et conditions

1.1 Principe

Dans le précédent chapitre, vous avez pu vous familiariser avec les expressions mettant en place des opérateurs, qu'ils soient de calcul, de comparaison (l'égalité) ou booléens. Ces opérateurs et expressions trouvent tout leur sens une fois utilisés dans des conditions (qu'on appelle aussi des branchements conditionnels). Une expression évaluée est ou vraie (le résultat est différent de zéro) ou fausse. Suivant ce résultat, l'algorithme va effectuer une action, ou une autre. C'est le principe de la condition.

Grâce aux opérateurs booléens, l'expression peut être composée : plusieurs expressions sont liées entre elles à l'aide d'un opérateur booléen, éventuellement regroupées avec des parenthèses pour en modifier la priorité.

```
(a=1 OU (b*3=6)) ET c>10
```

est une expression tout à fait valable. Celle-ci sera vraie si chacun de ses composants respecte les conditions imposées. Cette expression est vraie si a vaut 1 et c est supérieur à 10 ou si b vaut 2 ($2*3=6$) et c est supérieur à 10.

Reprenez l'algorithme du précédent chapitre qui calcule les deux résultats possibles d'une équation du second degré. L'énoncé simplifié disait que pour des raisons pratiques seul le cas où l'équation a deux solutions fonctionne. Autrement dit, l'algorithme n'est pas faux dans ce cas de figure, mais il est incomplet. Il manque des conditions pour tester la valeur du déterminant : celui-ci est-il positif, négatif ou nul ? Et dans ces cas, que faire et comment le faire ?

Imaginez un second algorithme permettant de se rendre d'un point A à un point B. Vous n'allez pas le faire ici réellement, car c'est quelque chose de très complexe sur un réseau routier important. De nombreux sites Internet vous proposent d'établir un trajet avec des indications. C'est le résultat qui est intéressant. Les indications sont simples : allez tout droit, tournez à droite au prochain carrefour, faites trois kilomètres et au rond-point prenez la troisième sortie direction B. Dans la plupart des cas, si vous suivez ce trajet vous arrivez à bon port. Mais quid des impondérables ? Par où allez-vous passer si la route à droite au prochain carrefour est devenue un sens interdit (cela arrive parfois, y compris avec un GPS, prudence) ou que des travaux empêchent de prendre la troisième sortie du rond-point ?

Reprenez le trajet : allez tout droit. Si au prochain carrefour la route à droite est en sens interdit : continuez tout droit puis prenez à droite au carrefour suivant puis à gauche sur deux kilomètres jusqu'au rond-point. Sinon : tournez à droite et faites trois kilomètres jusqu'au rond-point. Au rond-point, si la sortie vers B est libre, prenez cette sortie. Sinon, prenez vers C puis trois cents mètres plus loin tournez à droite vers B.

Ce petit parcours ne met pas uniquement en lumière la complexité d'un trajet en cas de détour, mais aussi les nombreuses conditions qui permettent d'établir un trajet en cas de problème. Si vous en possédez, certains logiciels de navigation par GPS disposent de possibilités d'itinéraire Bis, de trajectoire d'évitement sur telle section, ou encore pour éviter les sections à péage. Pouvez-vous maintenant imaginer le nombre d'expressions à évaluer dans tous ces cas de figure, en plus de la vitesse autorisée sur chaque route pour optimiser l'heure d'arrivée ?

1.2 Que tester ?

Les opérateurs s'appliquent sur quasiment tous les types de données, y compris les chaînes de caractères, tout au moins en pseudo-code algorithmique. Vous pouvez donc quasiment tout tester. Par tester, comprenez ici évaluer une expression qui est une condition. Une condition est donc le fait d'effectuer des tests pour, en fonction du résultat de ceux-ci, effectuer certaines actions ou d'autres.

Une condition est donc une affirmation : l'algorithme et le programme ensuite détermineront si celle-ci est vraie, ou fausse.

Une condition retournant VRAI ou FAUX a comme résultat un **booléen**.

En règle générale, une condition est une comparaison même si en programmation une condition peut être décrite par une simple variable (ou même une affectation) par exemple. Pour rappel, une comparaison est une expression composée de trois éléments :

- une première valeur : variable ou scalaire
- un opérateur de comparaison
- une seconde valeur : variable ou scalaire.

Les opérateurs de comparaison sont :

- L'égalité : =
- La différence : != ou <>
- Inférieur : <
- Inférieur ou égal : <=
- Supérieur : >
- Supérieur ou égal : >=

Le pseudo-code algorithmique n'interdit pas de comparer des chaînes de caractères. Évidemment, vous prendrez soin de ne pas mélanger les torchons et les serviettes, en ne comparant que les variables de types compatibles. Dans une condition une expression, quel que soit le résultat de celle-ci, sera toujours évaluée comme étant soit vraie, soit fausse.

■ Remarque

L'opérateur d'affectation peut aussi être utilisé dans une condition. Dans ce cas, si vous affectez 0 à une variable, l'expression sera fausse et si vous affectez n'importe quelle autre valeur, elle sera vraie.

En langage courant, il vous arrive de dire "choisissez un nombre entre 1 et 10". En mathématique, vous écrivez cela comme ceci :

$$1 \leq \text{nombre} \leq 10$$

Si vous écrivez ceci dans votre algorithme, attendez-vous à des résultats surprenants le jour où vous allez le convertir en véritable programme. En effet les opérateurs de comparaison ont une priorité, ce que vous savez déjà, mais l'expression qu'ils composent est aussi souvent évaluée de gauche à droite. Si la variable nombre contient la valeur 15, voici ce qui se passe :

- L'expression $1 = 15$ est évaluée : elle est vraie.
- Et ensuite ? Tout va dépendre du langage, l'expression suivante $\text{vrai} = 10$ peut être vraie aussi.
- Le résultat est épouvantable : la condition est vérifiée et le code correspondant va être exécuté !

Vous devez donc proscrire cette forme d'expression. Voici celles qui conviennent dans ce cas :

```
nombre >= 1 ET nombre <= 10
```

Ou encore

```
1 <= nombre ET nombre <= 10
```

1.3 Tests SI

1.3.1 Forme simple

Il n'y a, en algorithmique, qu'une seule instruction de test, "**Si**", qui prend cependant deux formes : une simple et une complexe. Le test SI permet d'exécuter du code si la condition (la ou les expressions qui la composent) est vraie.

La forme simple est la suivante :

```
Si booléen Alors
  Bloc d'instructions
FinSi
```

Notez ici que le booléen est la condition. Comme indiqué précédemment, la condition peut aussi être représentée par une seule variable. Si elle contient 0, elle représente le booléen FAUX, sinon le booléen VRAI.

Que se passe-t-il si la condition est vraie ? Le bloc d'instructions situé après le "**Alors**" est exécuté. Sa taille (le nombre d'instructions) n'a aucune importance : de une ligne à n lignes, sans limite. Dans le cas contraire, le programme continue à l'instruction suivant le "**FinSi**". L'exemple suivant montre comment obtenir la valeur absolue d'un nombre avec cette méthode.

```
PROGRAMME ABS
VAR
  Nombre :entier
DEBUT
  nombre←-15
  Si nombre<0 Alors
    nombre←-nombre
  FinSi
  Afficher nombre
FIN
```

En PHP, c'est le "if" qui doit être utilisé avec l'expression booléenne entre parenthèses. La syntaxe est celle-ci :

```
if(boolean) { /*code */ }
```

Si le code PHP ne tient que sur une ligne, les accolades peuvent être supprimées, comme dans l'exemple de la valeur absolue. Cet exemple montre une seconde possibilité par les mécanismes offerts par les bibliothèques de fonction de PHP.

```
<html>
  <head><meta/>
    <title>ABS</title>
  </head>
  <body>
    <?php
      $nombre=-15;
```

```
if($nombre<0) $nombre=-$nombre;
echo $nombre;

echo "<br />";
// seconde possibilité
$nombre2=-32;
$nombre2=abs($nombre2);
echo $nombre2;
?>
</body>
</html>
```

1.3.2 Forme complexe

La forme complexe n'a de complexe que le nom. Il y a des cas où il faut exécuter quelques instructions si la condition est fausse sans vouloir passer tout de suite à l'instruction située après le FinSi. Dans ce cas, utilisez la forme suivante :

```
Si booléen Alors
    Bloc d'instructions
Sinon
    Bloc d'instructions
FinSi
```

Si la condition est vraie, le bloc d'instructions situé après le Alors est exécuté. Ceci ne diffère pas du tout de la première forme. Cependant, si la condition est fausse, cette fois c'est le bloc d'instructions situé après le Sinon qui est exécuté. Ensuite, le programme reprend le cours normal de son exécution après le FinSi.

Notez que vous auriez pu très bien faire un équivalent de la forme complexe en utilisant deux formes simples : la première avec la condition vraie, la seconde avec la négation de cette condition. Mais ce n'est pas très joli, même si c'est correct. Retenez que :

- Si dans une forme complexe, l'un des deux blocs d'instructions est vide, alors transformez-la en forme simple : modifiez la condition en conséquence.
- Laisser un bloc d'instructions vide dans une forme complexe n'est pas conseillé : c'est une grosse maladresse de programmation qui peut être facilement évitée, c'est un programme sale. Cependant, certains langages l'autorisent, ce n'est pas une erreur ni même une faute lourde, mais ce n'est pas une raison...