Cours : Initiation à l'Algorithme avec Exercices

1 Introduction

Un algorithme est une suite d'instructions précises et ordonnées permettant de résoudre un problème ou d'accomplir une tâche. En informatique, les algorithmes sont la base de tout programme, comparables à une recette de cuisine : une série d'étapes pour atteindre un objectif.

1.1 Objectifs du cours

— Comprendre ce qu'est un algorithme et son rôle en informatique.
— Apprendre à décomposer un problème en étapes simples.
— Découvrir les outils pour représenter un algorithme (pseudo-code, organigramme).
— Pratiquer la conception d'algorithmes à travers des exercices variés.

2 Qu'est-ce qu'un algorithme ?

Un algorithme est une méthode systématique pour résoudre un problème. Il doit être :
— **Fini** : Il se termine après un nombre limité d'étapes.
— **Précis** : Chaque instruction est claire et non ambiguë.
— **Efficace** : Il utilise les ressources (temps, mémoire) de manière raisonnable.

2.1 Exemple quotidien

Pour préparer un café :
1. Remplir la cafetière d'eau.
2. Ajouter du café moulu dans le filtre.
3. Allumer la cafetière.
4. Attendre que le café soit prêt.
5. Servir le café.

3 Caractéristiques d'un bon algorithme

Un algorithme doit respecter ces critères :
1. **Entrées** : Les données nécessaires pour résoudre le problème.
2. **Sorties** : Le résultat attendu.
3. **Définition claire** : Chaque étape est compréhensible.
4. **Finitude** : L'algorithme s'arrête après un nombre fini d'étapes.
5. **Efficacité** : Il minimise le temps et les ressources.

4 Représentation d'un algorithme

Les algorithmes peuvent être décrits de plusieurs façons :

— **Langage naturel** : Description en phrases simples.

— **Pseudo-code** : Langage intermédiaire, structuré.

— **Organigramme** : Représentation graphique avec des symboles.

4.1 Exemple en pseudo-code

Problème : Calculer la somme de deux nombres.

```
DEBUT
  LIRE nombre1, nombre2
  somme <- nombre1 + nombre2
  AFFICHER somme
FIN
```

5 Concepts de base

5.1 Variables

Une variable est une zone de mémoire qui stocke une valeur. Exemple : `compteur <- 0`.

5.2 Instructions

— **Affectation** : `x <- 10`.

— **Entrée/Sortie** : `LIRE x`, `AFFICHER x`.

— **Calculs** : `y <- x * 2`.

5.3 Structures de contrôle

1. **Séquencielle** : Instructions exécutées dans l'ordre.
2. **Conditionnelle** :

```
SI condition ALORS
  Faire ceci
SINON
  Faire cela
FIN SI
```

3. **Itérative** :
   — Boucle **POUR** :

```
POUR i DE 1 A 5 FAIRE
  AFFICHER i
FIN POUR
```

   — Boucle **TANT QUE** :

```
TANT QUE x < 5 FAIRE
  x <- x + 1
  AFFICHER x
FIN TANT QUE
```

6 Étapes pour concevoir un algorithme

1. Comprendre le problème.
2. Décomposer en sous-étapes.
3. Écrire l'algorithme en pseudo-code.
4. Tester avec des exemples.
5. Optimiser si nécessaire.

7 Exercices pratiques

Cette section propose des exercices pour pratiquer la conception d'algorithmes, avec des solutions détaillées.

7.1 Exercice 1 : Calculer le périmètre d'un rectangle

Écrire un algorithme qui lit la longueur et la largeur dun rectangle et affiche son périmètre.
Solution :

```
DEBUT
  LIRE longueur, largeur
  perimetre <- 2 * (longueur + largeur)
  AFFICHER "Le périmètre du rectangle est ", perimetre
FIN
```

7.2 Exercice 2 : Déterminer si un nombre est divisible par 3

Écrire un algorithme qui lit un nombre et affiche sil est divisible par 3.
Solution :

```
DEBUT
  LIRE nombre
  SI nombre MOD 3 = 0 ALORS
    AFFICHER "Le nombre est divisible par 3"
  SINON
    AFFICHER "Le nombre n'est pas divisible par 3"
  FIN SI
FIN
```

7.3 Exercice 3 : Afficher les nombres pairs jusqu'à 10

Écrire un algorithme qui affiche tous les nombres pairs de 2 à 10.
Solution :

```
DEBUT
  POUR i DE 2 A 10 AVEC PAS DE 2 FAIRE
    AFFICHER i
  FIN POUR
FIN
```

7.4 Exercice 4 : Calculer la factorielle d'un nombre

Écrire un algorithme qui lit un nombre entier positif n et calcule sa factorielle ($n! = 1 \times 2 \times \cdots \times n$).

Solution :

```
DEBUT
  LIRE n
  SI n < 0 ALORS
    AFFICHER "Erreur : le nombre doit être positif"
  SINON
    factorielle <- 1
    POUR i DE 1 A n FAIRE
      factorielle <- factorielle * i
    FIN POUR
    AFFICHER "La factorielle de ", n, " est ", factorielle
  FIN SI
FIN
```

7.5 Exercice 5 : Vérifier si un nombre est premier

Écrire un algorithme qui lit un nombre entier positif et détermine sil est premier (divisible uniquement par 1 et lui-même).

Solution :

```
DEBUT
  LIRE nombre
  SI nombre <= 1 ALORS
    AFFICHER "Le nombre n'est pas premier"
  SINON
    est_premier <- VRAI
    POUR i DE 2 A nombre - 1 FAIRE
      SI nombre MOD i = 0 ALORS
        est_premier <- FAUX
      FIN SI
    FIN POUR
    SI est_premier ALORS
      AFFICHER "Le nombre est premier"
    SINON
      AFFICHER "Le nombre n'est pas premier"
    FIN SI
  FIN SI
FIN
```

7.6 Exercice 6 : Inverser un nombre à deux chiffres

Écrire un algorithme qui lit un nombre à deux chiffres (entre 10 et 99) et affiche son inverse (ex. : 23 devient 32).

Solution :

```
DEBUT
  LIRE nombre
  SI nombre >= 10 ET nombre <= 99 ALORS
```

```
    unite <- nombre MOD 10
    dizaine <- nombre DIV 10
    inverse <- unite * 10 + dizaine
    AFFICHER "L'inverse de ", nombre, " est ", inverse
  SINON
    AFFICHER "Erreur : le nombre doit avoir deux chiffres"
  FIN SI
FIN
```

7.7 Exercice 7 : Compter les voyelles dans une chaîne

Écrire un algorithme qui lit une chaîne de caractères et compte le nombre de voyelles (a, e, i, o, u).

Solution :

```
DEBUT
  LIRE chaine
  compteur <- 0
  POUR chaque caractere DANS chaine FAIRE
    SI caractere IN ('a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U')
        ALORS
      compteur <- compteur + 1
    FIN SI
  FIN POUR
  AFFICHER "Le nombre de voyelles est ", compteur
FIN
```

8 Conclusion

Les algorithmes sont essentiels pour structurer la résolution de problèmes en informatique. En pratiquant ces exercices, vous développerez une pensée logique et serez prêt à coder dans des langages comme Python ou JavaScript.

8.1 Conseils pour progresser

- Résoudre les exercices sans regarder les solutions d'abord.
- Tester vos algorithmes avec des cas limites (zéro, nombres négatifs).
- Explorer des problèmes plus complexes sur des plateformes comme LeetCode.

9 Ressources supplémentaires

- **Livres** : "Introduction to Algorithms" de Cormen.
- **Sites** : Codecademy, LeetCode, Khan Academy.
- **Outils** : Visual Studio Code pour écrire et tester vos algorithmes.