L1 Informatique

Algorithmique et Structure de Données 1

Dr Meriem BENADDA
Universite A. Mira - Béjaïa

© Octobre 2025

Chapitre 1 : Les éléments de base de l'algorithmique

Que sont les algorithmes ? Pourquoi l'étude des algorithmes est-elle utile ? Quel est le rôle des algorithmes par rapport aux autres technologies utilisées dans les ordinateurs ?

Ce chapitre répondra à ces questions.

1.1 Introduction

L'algorithmique est une discipline fondamentale en informatique, c'est la science qui étudie la manière de résoudre un problème en décrivant, étape par étape, la suite d'instructions nécessaires pour arriver à la solution. Avant d'écrire un programme dans un langage informatique, il est essentiel de passer par la phase algorithmique pour concevoir une méthode claire, logique, compréhensible et exécutable par un ordinateur [1].

1.2 Notion d'algorithme

1.2.1 Origine

Le terme algorithme trouve son origine dans le nom du mathématicien perse Muḥammad ibn Mūsā al-Khwārizmī (IX^{e} siècle), considéré comme l'un des fondateurs de l'algèbre.

Lors de la traduction en latin de ses travaux au XII^{e} siècle, son nom fut transcrit en *Algoritmi*, puis utilisé pour désigner les procédés de calcul décrits dans ses écrits.

Avec le temps, le mot a évolué en *algoritmus*, puis en français moderne en algorithme.

1.2.2 Définition

Un algorithme est une suite finie et ordonnée d'opérations (instructions) permettant de résoudre un problème. Un algorithme doit respecter certaines caractéristiques fondamentales [1],[2]:

- **Finitude** : Un algorithme doit avoir un nombre fini d'étapes.
- **Précision** : Chaque instruction doit être définie sans ambiguïté.

Copyright © BENADDA Meriem

- **Efficacité** : Un algorithme doit aboutir à un résultat (résultat recherché), en un temps raisonnable.
- **Entrées et sorties** : Un algorithme doit avoir des données en entrée et produire des résultats en sortie.

1.3 Langage de programmation

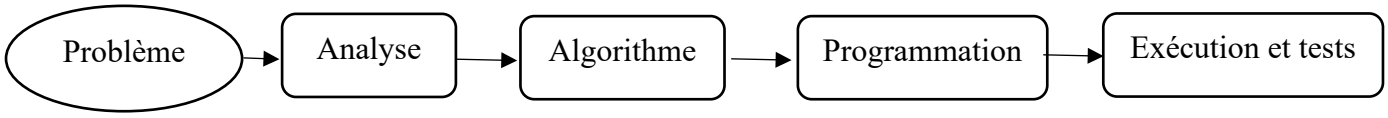
Un langage de programmation est un moyen formel permettant d'écrire des algorithmes sous une forme compréhensible par une machine. Il existe plusieurs types de langages, notamment [3]:

- **Langages de bas niveau** : proches du langage machine (ex. Assembleur).
- **Langages de haut niveau** : plus compréhensibles pour les humains (ex. Python, C, Java).

Un programme informatique est une implémentation concrète d'un algorithme à l'aide d'un langage de programmation.

1.4 Étapes de résolution d'un problème

La résolution d'un problème informatique passe par plusieurs étapes clés [4]:



1. **Analyse du problème** : Cette étape consiste à bien comprendre le problème posé avant de chercher une solution. On se pose des questions essentielles telles que : **Quelles sont les données disponibles en entrée ? Quels résultats sont attendus en sortie ? Quelles sont les contraintes ou conditions à respecter ? Quels cas particuliers faut-il prévoir ?** Cette phase permet d'identifier les entrées, les sorties et contraintes du problème tout en définissant clairement les objectifs du programme et en évitant les ambiguïtés qui pourraient compliquer les étapes suivantes.

2. **Conception de l'algorithme** : Consiste à élaborer une suite logique d'instructions en suivant une structure bien précise.

3. **Programmation** : Consiste à traduire l'algorithme à l'aide d'un langage de programmation.

4. **Exécution et tests** : Consiste à valider le programme en testant différents cas de figure.

À partir de la section suivante, chaque notion algorithmique sera accompagnée de son équivalent en langage C, qui constitue le langage de programmation utilisé dans les séances de travaux pratiques.

1.5 Structure d'un algorithme

Un algorithme est généralement présenté sous une forme structurée afin d'être clair, lisible et compréhensible. On adopte une organisation standard composée de trois parties principales :

Copyright © BENADDA Meriem

```
Algorithme id_Algorithme ; // En-tête

Variables <Déclaration des variables> ;
Constantes < Déclaration des constantes> ;        // Déclaration

Début
   <instruction_1> ;
      .                          // Corps de l'algorithme
      .
      .
   <instruction_N> ;

Fin.
```

a. **En-tête :**
Contient l'identificateur de l'algorithme, c'est un nom qui identifie l'algorithme et qui doit respecter certaines règles que nous aborderons plus loin dans le cours.

Exemple :

Algorithme CalculSomme ;

b. **Déclaration :**
Partie où l'on définit les variables, leurs types et éventuellement les constantes.

c. **Corps (ou bloc d'instructions) :**
Partie principale qui décrit les instructions à exécuter étape par étape, elle est délimitée par les mots-clés **Début** et **Fin.**

En C :

```
#include<stdio.h>

int main() {

      //instructions

return 0;
}
```

1.6 Notion de variable et de constante

1.6.1 Variable

Une variable est une zone mémoire identifiée par un nom (identificateur) et contenant une valeur modifiable durant l'exécution du programme.

1.6.2 Déclaration des variables

Les variables sont déclarées en utilisant le mot-clé `Variables` (Une seule fois pour toutes les variables), suivi de l'identificateur et de son type. La forme générale est :

```
Variables id_variable : type ;
```

Copyright © BENADDA Meriem

En C :

```
type id_variable;
```

Exemple : **En C**

```
Variables x : entier;    →    int x ;
          y : réel ;          float y ;
```

Les variables de même type peuvent être déclarées ensemble :

Exemple : **En C**

```
Variables a,b,c : entier; →    int a,b,c;
```

1.6.3 Constante

Une constante est une valeur fixe qui ne peut pas être modifiée après sa déclaration. Son objectif est d'éviter d'utiliser une valeur d'une manière direct. Imaginons qu'un algorithme utilise la valeur 3.14 une quinzaine de fois et qu'on veut modifier cette valeur par une autre valeur plus précise à savoir 3.14159. Dans ce cas on est amené à modifier toutes les occurrences de 3.14. Par contre, si on utilise une constante PI = 3.14 on ne la modifiera qu'une seule fois.

1.6.2 Déclaration des constantes

Les constantes sont déclarées en utilisant le mot-clé `Constantes` (Une seule fois pour toutes les constantes pour la déclaration algorithmique), suivi de l'identificateur et de la valeur de la constante. La forme générale est :

```
Constantes id_constante = valeur ;
```

En C :

Exemple : **En C :**

```
Constantes PI = 3.1415 ;   →   const float PI = 3.1415 ;
```

Remarque : En C, chaque déclaration de constante doit être précédée par le mot-clé **const** sauf si les constantes ont le même type. **Exemple** `const int a=15 , b=10 , c = 20 ;`

1.7 Notion d'identificateur

Un identificateur est le nom attribué à une variable, une constante ou une fonction et même à l'algorithme. Il doit respecter certaines règles :

- Ne peut commencer que par un caractère alphabétique (lettre) ou un underscore (_).
- Ne peut contenir que des caractères alphanumériques (chiffres et lettres) et underscore (_), il ne peut, donc pas contenir des caractères spéciaux ou de ponctuation tels que #, (l'espace), $, &, =, <,.. ou les lettres grecs comme π, β, Ω, etc.
- Doit être unique.
- Dans un programme, un identificateur ne peut pas être un mot-clé du langage de programmation. **Exemple** : En C, on ne peut pas utiliser `for`, `while`, `if`, `until`, etc.

Copyright © BENADDA Meriem

1.8 Types de base en algorithmique

Les types de base sont les catégories fondamentales de données utilisées en algorithmique (programmation). Voici les principaux types avec leur représentation en mémoire :

- **Entier (int, long)** : Représente l'ensemble Z des entiers. Exemple : -3, 0, 20075 ..
- **Réel (float, double)** : Représente l'ensemble R des réels. Exemple : 3.14, -0.5, 123.789..
- **Caractère (char)** : Peut être tous les symboles que nous trouvons sur le clavier d'un ordinateur : espace, lettres, chiffres, signes de ponctuation et caractères spéciaux (#, @, &, etc.).
- **Chaîne de caractères (char id_variable[n], n étant le nombre de caractères)** : suite de caractères, stockée en mémoire sous forme de tableau de charactères.
- **Booléen (bool)** : Une donnée de type booléen peut avoir deux valeur « vrai » ou « faux » (true, false). Stocké sur 1 bit théoriquement (Le bit (abréviation de *binary digit*) est la plus petite unité d'information utilisée en informatique. Il ne peut prendre que deux valeurs 0 ou 1, un octet ou Byte (en anglais) est une suite de 8 bits).

  Exemples :

Algorithmique		**En C**
Variables nom : chaîne de caractères [20] ;	→	`char nom[20];`
Constantes nom = "BENADDA" ;	→	`const char nom[10] = "BENADDA" ;`

Une fois les variables et les constantes introduites, nous pouvons aborder la notion d'expression, qui représente la manière dont les données sont combinées et traitées pour produire de nouvelles valeurs au sein d'un algorithme.

1.9 Expressions

1.9.1 Définition

Une expression est une combinaison de constantes, de variables et d'opérateurs (arithmétiques, relationnels, logiques, etc.) qui peut être évaluée pour produire une valeur.
Les expressions constituent les éléments essentiels des instructions de calcul, de comparaison ou de décision d'un algorithme. En algorithmique, les expressions peuvent être arithmétiques ou logique (booléennes).

1.9.2 Expressions arithmétiques

Une expression arithmétique est constituée d'opérandes numériques (valeurs, variables, constantes) reliés par des opérateurs arithmétiques.

Copyright © BENADDA Meriem

Opération	Algorithmique	En C
Addition	+	+
Soustraction	-	-
Multiplication	*	*
Division réelle	/	/
Division entière	div	/
Modulo	mod	%

Le modulo est le reste de la division entière.

1.9.2 Expressions relationnelles

Une expression relationnelle utilise des opérateurs relationnels (ou de comparaison). Le résultat est booléen (vrai ou faux).

Opération	Algorithmique	En C
Égalité	=	==
Différence	≠	!=
Supériorité stricte	>	>
Infériorité stricte	<	<
Supérieur ou égal	>=	>=
Inférieur ou égal	<=	<=

1.9.3 Expressions logiques (booléennes)

Une expression logique est une combinaison de variables de type booléen et d'opérateurs booléens. Elle peut combiner des expressions relationnelle. Le résultat est booléen.

Opération	Algorithmique	En C
Négation	Non	!
Conjonction	Et	&&
Disjonction	Ou	||

1.9.4 Evaluation des expressions

L'évaluation d'une expression consiste à calculer sa valeur en appliquant l'ordre de priorité des opérateurs.

En algorithmique comme en langage C, certains opérateurs sont prioritaires sur d'autres, ce qui influence le résultat final d'un calcul.

1.9.4.1 Ordre de priorité des opérateurs (du plus prioritaire au moins prioritaire)

Copyright © BENADDA Meriem

Lors de l'évaluation d'une expression, les opérateurs ne sont pas exécutés dans l'ordre d'écriture, mais selon leur niveau de priorité. Voici cet ordre, du plus fort au plus faible :

1. **Parenthèses** : `()`
   Permettent de forcer l'ordre d'évaluation**.** Exemple : `(a + b) * c`

2. **Appels de fonctions**

   Les fonctions sont évaluées avant d'être utilisées dans un calcul.

   Exemple : `x + y * 3 - sqrt(a + b)` → la fonction `sqrt()` est exécutée avant toute autre opération.

3. Opérateurs unaires **:** - (signe négatif), Non (négation logique)
4. Multiplication, division et modulo
5. Addition et soustraction
6. Comparaisons relationnelles
7. Tests d'égalité ou de différence
8. ET logique
9. OU logique

Exemples de fonctions courantes en algorithmique et en langage C

Fonction (algorithmique)	**Rôle/ Description**	**Équivalent en langage C**	**Bibliothèque C nécessaire**	**Exemple d'utilisation en C**
abs(x)	Donne la valeur absolue de x	abs(x)/ fabs(x)	<stdlib.h> / <math.h>	val = abs(-5); // 5
racine(x)	Calcule la racine carrée de x	sqrt(x)	<math.h>	r = sqrt(9); // 3
puiss(x, y)	Calcule x à la puissance y	pow(x, y)	<math.h>	p = pow(2, 3); // 8
ent(x)	Donne la partie entière de x	floor(x) / ceil(x)	<math.h>	e = floor(3.7); // 3 e = ceil(3.7); // 4
arrondi(x)	Arrondit x à l'entier le plus proche	round(x)	<math.h>	a = round(3.6); // 4
alea(n)	Génère un nombre aléatoire inférieur à n	rand() % n	<stdlib.h>	x = rand() % 10;
longueur(chaine)	Donne la longueur d'une chaîne de caractères	strlen(chaine)	<string.h>	n = strlen("test"); // 4
majuscule(chaine)	Convertit une chaîne en majuscules	strupr(chaine)	<string.h>	strupr(nom);
minuscule(chaine)	Convertit une chaîne en minuscules	strlwr(chaine)	<string.h>	strlwr(prenom);

Remarques importantes :

- Les opérateurs d'un même niveau de priorité sont évalués de gauche à droite.
- L'utilisation de parenthèses est recommandée pour éviter toute ambiguïté et clarifier la lecture du code.

Copyright © BENADDA Meriem

Après avoir établi l'ordre de priorité des opérateurs, il est nécessaire d'introduire la table de vérité, qui décrit la manière dont les expressions logiques sont évaluées selon les différentes valeurs possibles des variables booléennes.

1.9.4.2 Table de vérité

Une table de vérité indique le résultat (Vrai ou Faux) d'une expression logique pour toutes les combinaisons possibles des valeurs des variables.

A	B	Non A	A ET B	A OU B
Vrai	Faux	Faux	Faux	Vrai
Vrai	Vrai	Faux	Vrai	Vrai
Faux	Vrai	Vrai	Faux	Vrai
Faux	Faux	Vrai	Faux	Faux

Après avoir étudié les variables, les constantes et les expressions, il est maintenant nécessaire de comprendre comment elles sont utilisées dans un programme.

1.10 Instructions de base en algorithmique

Les instructions de base sont les briques élémentaires qui permettent d'écrire tout algorithme : elles permettent d'agir sur les données, de réaliser des calculs et de communiquer avec l'utilisateur. Elles regroupent essentiellement : l'instruction de lecture, l'instruction d'écriture et l'instruction d'affectation.

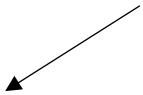
1.10.1 Instruction de lecture

La lecture est une instruction qui permet de saisir des données au clavier et de les stocker dans des variables pour être utilisées dans l'algorithme. Seules les variables peuvent être lues.

Syntaxe :

Lire (id_variable) ;

En C :



scanf("spécificateur de format", & id_variable);

Exemples :

```
scanf("%d", & id_variable); // pour un entier ou un booléen, on peut utiliser aussi %i

scanf("%f", & id_variable); // pour un réel

scanf("%c", & id_variable); // pour un caractère

scanf("%s",  id_variable); // pour une chaîne de caractères
```

Remarque : La lecture d'une chaîne de caractères se fait sans indiquer son adresse (sans &).

Copyright © BENADDA Meriem

1.10.2 Instruction d'écriture

L'écriture est une instruction qui permet d'afficher des messages ou des résultats à l'écran.

Syntaxe :

Ecrire ("Message à afficher") ; // affiche un message

Ecrire (id_variable) ; // affiche la valeur de la variable

Ecrire (expression) ; // affiche la valeur de l'expression

Ecrire (valeur) ; // affiche la valeur

En C :

printf("Message");

printf("spécificateur de format", id_variable);

printf("spécificateur de format", expression);

printf("spécificateur de format", valeur);

Remarque : On peut regrouper un message et une valeur dans la même écriture (en algorithmique et en C) en les séparant d'une virgule appelée virgule de concaténation.

Exemples :

Ecrire ("Le résultat est : ") ;
Ecrire (R);

→ Ecrire ("Le résultat est : ", R) ;

En C :

printf("Le résultat est : %d", id_variable);

Exemple :

Algorithme LectureAffichage ;
Variables x : entier ;
Début
  Écrire("Donner un entier :") ;
  Lire(x) ;
  Écrire("La valeur de x est :", x) ;
Fin.

En C :

```
#include <stdio.h>
int main() {
    int x;
    printf("Donner un entier : ");
    scanf("%d", &x);
    printf("La valeur de x est : %d\n", x);
    return 0;}
```

Copyright © BENADDA Meriem

1.10.3 Instruction d'affectation

L'affectation est une instruction qui permet de donner une valeur à une variable. Cette valeur peut être direct, celle d'une variable ou le résultat d'une expression.

Syntaxe :

id_variable ← valeur ; id_variable ← id_variable; id_variable ← expression ;

En C :

id_variable = valeur ; id_variable = id_variable; id_variable = expression ;

Remarques :

- Lorsque l'on donne une valeur pour la première fois à une variable en utilisant une affectation, cette opération s'appelle **l'initialisation**.

- Lorsque l'on augmente la valeur d'une variable numérique d'une certaine quantité, le plus souvent de 1, cette opération s'appelle **l'incrémentation**. Elle est très utilisée dans les boucles (Chapitre 3) et les comptages.

Exemple : Calcul de la moyenne d'une matière

Algorithme Moyenne ;
Variables TD, Ex, moy : réel ;
Début
Écrire (" Donner la note de TD :") ;
Lire(TD) ;
Écrire (" Donner la note de l'examen :") ;
Lire(Ex) ;
moy ← (TD * 2 + Ex *3) / 5 ;
Écrire (" Moyenne = ", moy) ;
Fin.

En C :

```
#include <stdio.h>
int main() {
    float TD, Ex, moy;
    printf("Donner la note de TD : ");
    scanf("%f", &TD);
    printf("Donner la note de l'examen : ");
    scanf("%f", &Ex);

    moy = (TD * 2 + Ex *3) / 5;
    printf(" Moyenne = %.2f\n ", moy);
    return 0;
}
```

Copyright © BENADDA Meriem

Avant d'aborder un nouveau chapitre, il est important d'apprendre à suivre pas à pas l'exécution d'un algorithme pour en comprendre le fonctionnement. C'est ce qu'on appelle une trace d'exécution.

1.11 Trace d'exécution

Une trace d'exécution est un tableau qui permet de visualiser le déroulement d'un algorithme étape par étape. Elle montre :

- L'évolution des valeurs des variables.
- Le contenu des affichages produits pendant l'exécution.

C'est un outil d'analyse et de vérification très utile pour s'assurer que l'algorithme fonctionne comme prévu.

La trace d'exécution se présente généralement sous forme d'un tableau à trois colonnes principales comme le montre le tableau suivant :

Instructions	**Variables**			**Affichage**
	V1	**…**	**Vn**	
Instruction1 ;				
…..				
InstructionN ;				

Exemple :

Algorithme Trace ;
Variables A, B, C : entier ;
Début
  A ← 3 ;
  B ← 5 ;
  C ← A * 2 + B ;
  Écrire("La somme est :", C) ;
Fin.

Trace d'exécution

Instructions	**Variables**			**Affichage**
	A	**B**	**C**	
A ← 3 ;	3	/	/	/
B ← 5 ;	3	5	/	/
C ← A * 2 + B ; C ← 3 * 2 + 5 = 11	3	5	11	/
Écrire("La somme est :", C) ;	3	5	11	La somme est : 11

Copyright © BENADDA Meriem

Conclusion

Dans ce premier chapitre, nous avons découvert les bases fondamentales de l'algorithmique: les variables, les constantes, les expressions et les instructions de base.

Néanmoins, afin de rendre un algorithme plus dynamique et capable d'adapter son comportement, il est indispensable d'utiliser des structures de contrôle permettant la prise de décision et la répétition d'actions.

Tests d'acquisition des connaissances

Q1. Qu'est-ce qu'un algorithme ?

Q2. Citez les principales étapes de résolution d'un problème en algorithmique.

Q3. Donnez un exemple de langage de programmation.

Q4. Quelle est la différence entre une variable et une constante ?

Q5. Quel type de base est utilisé pour stocker un nombre avec des décimales ?

Q6. Questions à choix multiple (QCM)

Q7. Un algorithme doit être :

a) Infini

b) Ambigu

c) Précis et fini

d) Exécuté uniquement sur papier

Q8. Quel est le rôle du langage de programmation ?

a) Traduire un algorithme en instructions compréhensibles par la machine

b) Remplacer l'algorithme

c) Tester uniquement le programme

d) Stocker des données

Q9. Quelle est la représentation mémoire d'un booléen ?

Copyright © BENADDA Meriem

a) 1 bit

b) 1 octet

c) 2 octets

d) 4 octets

Q10. Un identificateur peut commencer par :

a) Un chiffre

b) Un espace

c) Une lettre

d) Un caractère spécial (@, #, etc.)

Q11. Quel type de base est utilisé pour stocker une seule lettre ?

a) Entier

b) Booléen

c) Caractère

d) Réel

Q12. Classez les opérateurs suivants selon leur priorité : +, *, &&, <, ().

Q13. Dans scanf("%d", &x); que représente le symbole & ?

Q14. Qu'est-ce qu'une trace d'exécution et à quoi sert-elle ?

Copyright © BENADDA Meriem