

Université Abderrahmane MIRA - Bejaia

Faculté de Technologie

Département Technologie



Notion d'algorithme et de programme

Dr. A. OUARET



ahmed.ouaret@univ-bejaia.dz

Laboratoire de Technologie Industrielle et de l'Information,
Faculté de Technologie
Université de Bejaia, 06000 Bejaia, Algérie



Objectif du chapitre

À l'issue de ce chapitre, l'apprenant sera capable de :

1

Comprendre les notions de base de l'informatique et les principes fondamentaux de la programmation.

2

Maîtriser la structure d'un programme en langage C ainsi que les types de données et les variables.

3

Utiliser les opérations d'entrées/sorties et les expressions pour traiter l'information.

4

Mettre en œuvre les structures de contrôle conditionnelles et itératives pour résoudre des problèmes simples.

Concept d'un algorithme

Le mot « **Algorithme** » est inventé par le mathématicien « **ALKHAWARISMI** ».

Un Algorithme est l'**énoncé d'une séquence d'actions primitives** réalisant un traitement. Il décrit le plan ou les séquences d'actions de **résolution d'un problème donné**.



ALKHAWARISMI

En résumé

Un algorithme représente une solution à un problème, exprimée à travers une série d'instructions qui manipulent des données dans un ordre bien défini.

Problème



Série d'instructions



Données manipulées



Solution

Traduction d'un algorithme en programme

Une fois qu'un algorithme a été conçu, **quelle que soit la langue naturelle utilisée** (français, anglais, arabe, etc.), il doit être **traduit dans un langage de programmation** pour devenir un **programme exécutable**. Ce processus de traduction comprend :

Les trois étapes

1

Rédaction

La **rédaction du code source** dans le langage de programmation choisi,



2

Compilation

La **compilation** (traduction) de ce code source en instructions compréhensibles par la machine,



3

Exécution

L'**exécution** du programme obtenu sur un ordinateur.

C

Tout au long de ce cours, le langage de programmation utilisé sera le **langage C**.

1

Code::Blocks

2

MyC / MyCompiler

3

OnlineGDB



MyCompiler

Compilateur en ligne pour écrire et exécuter des programmes.

```

1 #include <stdio.h>
2 int main() {
3     printf("Bonjour, monde !\n");
4     return 0;
5 }
6
Output
Compilation successful.
Bonjour, monde !
Process finished with exit code 0
    
```

Éditeur de code

Écrivez et modifiez votre code C directement dans le navigateur.

Compilation / Exécution

Lancez la compilation et l'exécution de votre programme en un clic.

Console de sortie

Affiche les messages de compilation et le résultat de l'exécution.



Code::Blocks

Environnement de développement pour programmer en C et C++.

```

1 #include <stdio.h>
2
3 int main() {
4     printf("Bonjour, Code::Blocks!\n");
5     return 0;
6 }
    
```

```

----- Build: Debug in Hello (compiler: GNU GCC Compiler) -----
Checking for existence: C:\Users\...\Hello\bin\Debug\Hello.exe
Executing: "C:\Program Files\CodeBlocks\cb_console_runner.exe"
Process terminated with status 0 (0 minute(s), 0 second(s))
    
```

Éditeur de code

Écrivez et modifiez vos fichiers source (.c, .cpp) avec une interface complète.

Compilation / Exécution

Compilez et exécutez votre projet depuis l'interface.

Console de sortie

Affiche les messages de compilation et les résultats de votre programme.



OnlineGDB

Compilateur et débogueur en ligne pour le langage C et d'autres langages.

```

1 #include <stdio.h>
2
3 int main() {
4     printf("Bonjour, OnlineGDB!\n");
5     return 0;
6 }
    
```

```

Console
Compiling...
Compilation successful.
Running...
Bonjour, OnlineGDB!

Program finished with exit code 0
    
```

Éditeur de code

Écrivez et modifiez votre code en ligne depuis votre navigateur.

Compilation / Exécution

Compilez et exécutez votre programme en un clic.

Console de sortie

Affiche les messages de compilation et le résultat de l'exécution.

II.3. La démarche de résolution d'un problème



Figure 1 : Etapes d'analyse d'un problème

II.4. Représentation des Algorithmes

Un algorithme peut être représenté selon plusieurs notations. Les deux plus couramment utilisées dans l'enseignement de l'informatique sont :

Organigramme

Une représentation **graphique** utilisant des formes standardisées :



Ovale
début / fin



Rectangle
traitement



Losange
décision



Parallélogramme
entrée / sortie

Pseudocode

Une description **structurée** en français (ou en anglais) utilisant des mots-clés tels que :

SI

TANTQUE

POUR

DÉBUT

FIN

DÉBUT

// exemple

SI note $\geq$ 10 **ALORS**

Écrire("Admis")

FINSI

FIN

Représentation par pseudocode

Une notation intermédiaire entre le langage naturel et le langage de programmation



Mots-clés simples et conventionnels

Il permet de décrire un algorithme de manière structurée et lisible, sans se soucier de la syntaxe stricte d'un langage de programmation particulier.

DÉBUT

FIN

SI

SINON

TANT QUE

POUR

LIRE

ÉCRIRE



Remarque

Le pseudocode est **indépendant de tout langage** de programmation et peut être facilement traduit en **C**, **Python**, **Java**, etc. Il est généralement préféré à l'organigramme pour sa **concision** et sa **lisibilité**, en particulier pour les algorithmes complexes.

Structure d'un algorithme / programme

Un algorithme manipule des données : avant de les utiliser, il faut les **identifier** et les **déclarer** en utilisant les identificateurs. Un algorithme est constitué de **trois parties** :

1

Entête

Dans cette partie on déclare le nom de l'algorithme à travers un **identificateur**.

2

Déclarations

Dans cette partie on déclare toutes les données utilisées par l'algorithme.

3

Corps

Représente la séquence d'actions (instructions).

EXEMPLE

ENTÊTE

Algorithme Exo1 ;

DÉCLARATIONS

Variables

N : Entier ;

P, PT : Réels ;

CORPS

Début

Lire(N) ;

Lire(P) ;

PT $\leftarrow$ N $\times$ P ;

Écrire(PT) ;

Fin.

Structure d'un algorithme

Pour écrire un algorithme, la structure suivante doit être respectée :

Algorithme

ENTÊTE

```
Algorithme <nomAlgorithme> ;
```

DÉCLARATION

```
// Section de déclaration
```

```
Variable <identifiant> : <TypeDonnée> ;
```

```
Constante <identifiant> = <Valeur> ;
```

CORPS

Début

```
<Instruction 1> ;
```

```
<Instruction 2> ;
```

```
*****
```

```
Fin.
```

Langage C

ENTÊTE

```
#include <stdio.h>
```

```
int main ()
```

```
{
```

DÉCLARATION

```
// Section de déclaration
```

```
<TypeDonnée> <identifiant> ;
```

```
const <TypeDonnée> <identifiant> = <Valeur> ;
```

CORPS

```
<Instruction 1> ;
```

```
<Instruction 2> ;
```

```
*****
```

```
return 0 ;
```

```
}
```

REMARQUES

Langage C : à retenir

- 1 Pour commenter un programme en langage C, on écrit les commentaires entre

```
/* ... */.
```

Par exemple :

```
/* Ceci est un commentaire */
```

- 2 On peut aussi utiliser les **commentaires ligne** :

```
// Ceci est un commentaire ligne
```

- 3 En langage C, la **partie déclarative** peut être **avant** la fonction main, comme elle peut être **dans** la fonction main.

```
int a ; // avant main
int main ()
{
    int b ; // dans main
    ...
}
```

- 4 En langage C, la fonction **main** est le **point d'entrée** du programme : la première qui sera exécutée.

```
int main () ▶ départ
{
    ...
}
```

1

L'entête

L'entête

L'entête sert à **donner un nom à l'algorithme** en utilisant un **identificateur**. Ce dernier est précédé par le mot clé « **Algorithme** ».

Algorithme <identificateur> EXEMPLE DE FORME

Alors qu'est-ce qu'un identificateur ?

Identificateur

Un identificateur est une **chaîne de caractères** contenant uniquement des caractères alphanumériques (alphabétiques de [a-z] et [A-Z] et numériques [0-9]) et le tiret du 8 « _ » (trait souligné).

a - z
minuscules

A - Z
majuscules

0 - 9
chiffres

_
trait souligné

Règle : il doit commencer soit par une **lettre alphabétique**, soit par **_**.

Il identifie d'une manière unique :

un algorithme (ou un programme)

une variable

une constante

une procédure

une fonction

⚠ Mots réservés

Dans un langage de programmation donné, on n'a pas le droit d'utiliser les **mots réservés (mots clés)** du langage comme des identificateurs. Parmi les mots clés du langage C :

`auto` `break` `case` `char` `continue` `do` `double` `else` `extern` `float` `for` `goto` `if` `int` `long`
`return` `short` `sizeof` `static` `struct` `switch` `typedef` `union` `unsigned` `while` ...

Exemple :

Tableau II.3 : Exemples d'identificateurs valides et non valides

Identificateur valide	Identificateur non valide
✓ <code>a1</code>	✗ <code>x1 y</code> à cause du blanc ou de l'espace
✓ <code>a_1</code>	✗ <code>x1-y</code> à cause du signe - (tiret de 6)
✓ <code>A_1</code>	✗ <code>1xy</code> commence par un caractère numérique
✓ <code>x12y</code>	✗ <code>R?</code> à cause du caractère spécial (?)
✓ <code>x1_y</code>	✗ <code>Prix-HT</code> à cause du signe - (tiret de 6)
✓ <code>Hauteur</code>	✗ <code>Exo 04</code> à cause de l'espace
✓ <code>Prix_HT</code>	✗ <code>scanf</code> mot clé
✓ <code>Exo_04</code>	✗ <code>printf</code> mot clé

❖ **Remarque** Pour résumer, un identifiant valide doit :

- 1 **Commencer** par une lettre ou le tiret du 8 (_)
- 2 **Contenir uniquement** des lettres, des chiffres ou des tirets du 8
- 3 **Ne pas être** un mot-clé réservé du langage
- 4 **Ne contenir ni** espaces ni caractères spéciaux

2

La Partie déclarative

La partie déclarative

Dans la partie déclaration, on **déclare toutes les données** d'entrées et de sorties sous forme de **constantes** et de **variables**.

 **Constantes**

 **Variables**

II.5.2.1 Constantes

Les constantes sont des objets contenant des **valeurs non modifiables**. Les constantes sont **déclarées** comme suit :

En algorithme :

```
Constante <identifiant> = <valeur> ;
```

En langage C :

```
const <type> <identifiant> = <valeur> ;
```

Exemples de constantes

Algorithme

Constante PI = 3.14;

Constante réelle.

Constante MAX = 10;

Constante entière.

Constante cc = 'a';

Constante caractère.

Constante ss = 'algo';

Constante chaîne de caractère.

Constante b1 = true;

Constante booléenne.

Constante b2 = false;

Constante booléenne.

Langage C

const float PI = 3.14;

const int MAX = 10;

const char cc = 'a';

const char ss[10] = "algo";

const int b1 = 1;

const int b2 = 0;

Variables

Les variables sont des objets contenant des **valeurs pouvant être modifiées**. Les variables sont **déclarées** comme suit :

En algorithme :

```
Variable <identifiant> : <TypeDonnée> ;
```

En langage C :

```
<TypeDonnée> <identifiant> ;
```

5

Une variable appartient à **exactement un type** de données. Il existe **cinq types de données fondamentaux** :

Types de Données Fondamentaux et leur Déclaration en Algorithme et en C

Type de Données	Algorithme	Langage C	Exemples
Entier	x : Entier ;	int x;	..., -2, -1, 0, 1, 2, ...
Réel	x : Réel ;	float x; ou double x;	3.14, -0.5, 2.718
Caractère	c : Caractère ;	char c;	'A', '7', '@'
Chaîne de caractères	s : Chaîne ;	char s[50];	"Bonjour", "CS101"
Booléen	b : Booléen ;	int b; ou bool b;	VRAI / FAUX → 1 / 0

Exemples de variables

Algorithme	Langage C	Signification
Variable x : réel ;	float x ;	variable réelle
Variable n, m : entier ;	int n, m ;	deux variables entières
Variable s : chaîne de caractères ;	char s[100] ;	variable chaîne de caractères
Variable b1, b2, b3 : booléen ;	bool b1, b2, b3 ;	3 variables booléennes
Variable c1 : caractère ;	char c1 ;	variable caractère



Remarque

En plus des constantes et des variables, il est également possible de déclarer de **nouveaux types**, **fonctions** et **procédures**.

Types, spécificateurs et fonctions d'E/S

Comme vu précédemment, chaque type de données algorithmique possède un **équivalent direct en C**. Lors de l'implémentation d'un algorithme en C, il est également essentiel de connaître le **spécificateur de format** associé à chaque type, car ceux-ci sont requis par les fonctions d'entrée/sortie standard :



Correspondance Algorithme / Langage C

Algorithme	Langage C	Spécificateur
Entier	<code>int</code>	<code>%d</code>
Réel	<code>float</code>	<code>%f</code>
Booléen	<code>bool</code>	<code>%d</code>
Caractère	<code>char</code>	<code>%c</code>
Chaîne	<code>char[]</code>	<code>%s</code>



Remarque

Le type `bool` nécessite `#include <stdbool.h>`. **1** représente **VRAI** (TRUE) et **0** représente **FAUX** (FALSE).

3

Le corps de programme

Le corps de programme

Le corps d'un algorithme est constitué d'un ensemble d'**actions / instructions** ordonnées de manière **séquentielle et logique**. Ces instructions se divisent en **cinq types distincts** :

1 *Lecture*

Cette opération consiste à introduire des données dans l'algorithme. Une lecture consiste à donner une valeur arbitraire à une variable.

2 *Écriture*

Cette opération implique l'affichage de données. Elle permet d'afficher des résultats ou des messages.

3 *Affectation*

Elle permet de modifier les valeurs des variables en leur assignant de nouvelles valeurs.

Structures de contrôle

Ces structures permettent de modifier la séquentialité de l'algorithme. Elles sont utilisées pour sélectionner différents chemins d'exécution ou pour répéter un traitement.

4 *Structure de Test alternatif simple / double*

5 *Structures répétitives (itérative)*

La boucle **Pour**

La boucle **Tantque**

La boucle **Répéter**

; Dans le langage C, chaque instruction se termine par un **point-virgule**.

Types d'instructions

Toutes les instructions d'un programme sont écrites dans son **corps** (entre **Début** et **Fin** ; en C, entre { et } de la fonction **main**).

CORPS DU PROGRAMME

Début	→	{
Fin	→	}

On peut regrouper ces instructions en deux grands types :

1 Les instructions séquentielles simples

Entrées, sorties et affectation.

Entrées

Sorties

Affectation

2 Les structures de contrôle

Tests et boucles.

Tests

Boucles

Instructions d'Entrées/Sorties (Lecture / Écriture)

Entrées (Lecture)

Une instruction d'entrée nous permet, dans un programme, de **donner une valeur quelconque à une variable**. Ceci se réalise à travers l'opération de **lecture**.

La syntaxe et la sémantique d'une lecture est comme suit : **Tableau II.8**

Algorithme	Langage C	Signification
Lire(<id_var>);	<code>scanf ("%c", &<id_var>);</code>	Donner une valeur quelconque à la variable dont l'identifiant <i><id_var></i> .
Lire(<iv1>, <iv2>, ...);	<code>scanf (" %c %c ...", &<iv1>, &<iv2>, ...);</code>	Donner des valeurs aux variables <iv1>, <iv2>, <i>etc.</i>

❖ Remarque

Il est important de noter que l'instruction de lecture concerne **uniquement les variables** : on ne peut pas lire des constantes ou des valeurs. Lors de la lecture d'une variable dans un programme C, le programme **se bloque** en attendant la saisie d'une valeur via le clavier. Une fois la valeur saisie, on valide par la touche **entrée**, et le programme reprend l'exécution avec l'instruction suivante.

Lecture **scanf** → Programme bloqué → Saisie au clavier → Touche **entrée** → Instruction suivante

Exemple :

Tableau II.9 : Exemples d'entrées

Algorithme	Langage C
<pre>Lire (a, b, c) ; Lire (hauteur) ;</pre>	<pre>scanf("%d %d %d", &a, &b, &c); scanf("%f", &hauteur);</pre>

Sorties (Écriture)

Une instruction de sortie nous permet, dans un programme, d'**afficher un résultat** (données traitées) ou bien **un message** (chaîne de caractères). Ceci se réalise à travers l'opération d'**écriture**.

La syntaxe et la sémantique d'une écriture est comme suit : **Tableau II.10**

Algorithme	Langage C	Signification
<pre>Ecrire(<id_var> ou <id_const> ou <valeur>, ou <expression>)</pre>	<pre>printf("%f", <id_var> ou <id_const> ou <valeur> ou <expression>;</pre>	Afficher une valeur d'une variable, d'une constante, valeur immédiate ou calculée à travers une expression.

❖ Remarque

Il est à noter que l'instruction d'écriture **ne concerne pas uniquement les variables** : on peut écrire des **constantes**, des **valeurs** ou des **expressions** (arithmétiques ou logiques).

On peut écrire :

Variables

Constantes

Valeurs

Expressions

Exemple :

Tableau II.11 : Exemples de sorties

Algorithme	Langage C	Signification
écrire("Bonjour")	<code>printf ("Bonjour");</code>	Afficher le message Bonjour
écrire(a, b, c)	<code>printf("%f %f %f", a, b, c);</code>	Afficher les valeurs des variables a, b et c
écrire(5+2)	<code>printf("%f",5+2);</code>	Afficher le résultat de la somme de 5 et 2 : afficher 7

Instruction d'affectation

Une affectation consiste à **donner une valeur** (immédiate, constante, variable ou calculée à travers une expression) **à une variable**.

VALEUR
↓
VARIABLE

La syntaxe d'une affectation est : **Tableau II.12**

Algorithme	Langage C
<pre> <id_variable> ← <valeur> ou <id_variable> ou <expression> ; </pre>	<pre> <id_variable> = <valeur> ou <id_variable> ou <expression> ; </pre>

Une affectation possède **deux parties** : la partie gauche qui représente **toujours une variable**, et la partie droite qui peut être : **une valeur, variable ou une expression**. La condition qu'une affectation soit correcte est que : la partie droite doit être **du même type** (ou de type compatible) **avec la partie gauche**.

Partie gauche

Toujours une **variable**

Partie droite

Une **valeur**, une **variable** ou une **expression**

Exemple :

Tableau II.13 : Exemples d'affectation

Algorithme	Langage C	Signification
$a \leftarrow 5$	<code>a=5;</code>	Mettre la valeur 5 dans la variable a
$b \leftarrow a+5$	<code>b=a+5;</code>	Mettre la valeur de l'expression a+5 dans la variable b
$sup \leftarrow a>b$	<code>sup=a>b;</code>	a>b donne un résultat booléen, donc sup est une variable booléenne

Exercice

Énoncé → Algorithme → Programme C

Énoncé du problème → Algorithme → Programme en langage C

Écrire un **algorithme**, puis traduisez-le en **programme C**, pour chacun des problèmes suivants :

1 **Permuter** entre les deux variables X et Y ?

2 **Permuter** entre les trois variables X, Y et Z de telle sorte que la valeur de X soit dans Y, celle de Y dans Z et la valeur de Z dans X ?

3 Calculer le **quotient** et le **reste** de la division euclidienne de a par b ?

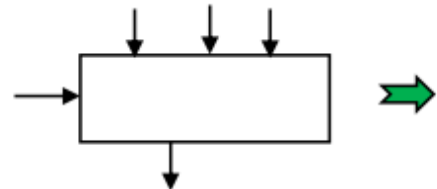
4 Calculer la **somme** de a et b et le **produit** de b et c ?

5 Calculer la **valeur absolue** et le **carré** d'un nombre réel ?

6 **Convertir** en octets un nombre donné en bits ?

7 Lire les notes de trois matières (N1 , N2 et N3) ensuite calculer et afficher leur **moyenne** M ?
Modifier l'algorithme dans le cas où des **coefficients** (C1 , C2 et C3) sont attribués aux trois matières.

Exo N°01 - TP N°02

Algorithme	Questions
Algorithme Exo1; Variables N : Entier ; P, PT : Réels ; Début // Entrées Écrire("Donner le nombre d'articles :") ; Lire(N) ; Écrire("Donner le prix unitaire :") ; Lire(P) ; // Traitement PT $\leftarrow$ N $\times$ P ; // Sortie Écrire ("Le prix total est : ", PT) ; Fin.	- Lister les variables de l'algorithme et préciser leurs types en C. - Traduire l'algorithme en programme C. - Compiler et exécuter le programme avec ($N=7$) et ($P=12.5$). - Écrire les instructions de lecture de (N) et (P) en C : - Avec un <code>scanf()</code> pour chaque variable. - Avec un seul <code>scanf()</code> pour les deux variables. - Écrire les instructions d'affichage du prix total PT avec <code>printf()</code> : - Afficher uniquement la valeur de PT. - Afficher un message explicatif suivi de PT. - Modifier la partie affichage de votre programme en remplaçant l'instruction actuelle par les quatre instructions suivantes : <code>printf("Prix total sans formatage :%f\n", PT);</code> <code>printf("Prix total avec formatage :%8.2f\n", PT);</code> <code>printf("Prix total avec formatage :%5.2f\n", PT);</code> <code>printf("Prix total avec formatage :%.2f\n", PT);</code> Compiler et exécuter le programme avec ($N=7$) et ($P=12.5$), que remarquez-vous concernant les résultats affichés ? Expliquez la différence entre les différents formats d'affichage utilisés. - Compléter le schéma suivant :  Entrée : Traitements : Sorties :

A hand in a dark suit jacket and white striped shirt cuff points its index finger towards a central white card. The card is surrounded by several other blue cards of various shapes and sizes, some overlapping. The background is a dark blue gradient.

Merci