Ecole Normale Supérieure de l'Enseignement Technique de Mohammedia
Université Hassan II de Casablanca

ENSET

المدرسة العليا لأساتذة التعليم التقني المحمدية
جامعة الحسن الثاني بالدار البيضاء

Département Mathématiques & Informatique

Algorithmique & Programmation

En Langage C

Cours TD TP

Objectifs

- **Comprendre les bases algorithmiques et Apprendre à utiliser les techniques de résolutions de problèmes informatiques :**
  - **Analyse descendante (abstraction successive),**
  - **Paradigme réduire/diviser pour régner,**
  - **Formulation récursive/itérative,**
  - **Structure de contrôle, Procédure/Fonction,**
  - **les structures de données : linked lists, stacks, queues, sets, hash tables, trees, heaps, priority queues, and graphs,**
- **Apprendre les techniques d'analyse d'algorithmes :**
  - **Efficacité, abstraction, réutilisation**
- **Apprendre à formuler des algorithmes et à les coder en langage C**
- **Apprendre à élaborer des applications programmée en langage C**

Sommaire

Chap. I : Introduction à l'algorithmique

- Notion de Système Informatique (SI)
  - Fonctions d'un SI
  - Information
  - Eléments matériels,
  - Eléments logiciels
- Codage binaire de l'information
- Notions d'algorithmique et d'algorithme
- Notion de Programme, compilateur/interpréteur
- Démarche méthodologique : Abstraction, Décomposition
- Exemples d'algorithmes
- Structure générale d'un algorithme

Sommaire

Chap. II : Les Concepts de base

- Les Types de données,
- Notion d'objets(Constantes et de variables)
- Notion d'instructions primitives
- Notion d'opérateurs et d'expressions
- Les Structures de contrôle
  - Séquence,
  - Test,
  - Boucles

Sommaire

Chap. III : Les fonctions et procédures

- Notion de modularité
- Les Fonctions et procédures
- La récursivité :
  - Récursivité terminale/centrale
  - Pile d'exécution
  - Test d'arrêt

Sommaire

Chap. IV : Les structures de données linéaires

- Notion de types abstraits de données
- Pointeurs et allocation dynamique de mémoire
- Les Tableaux – algorithmes sur les tableaux
  - Tri, recherche, …
- Les Listes
- Les Piles
- Les Files

Sommaire

Chap. V : Les fichiers

- Définition et Classification
- manipulation des fichiers
  - Cas des fichiers textes
  - Cas des fichiers binaires
- Applications

I. Notion de système informatique

1. Définitions

Informatique :

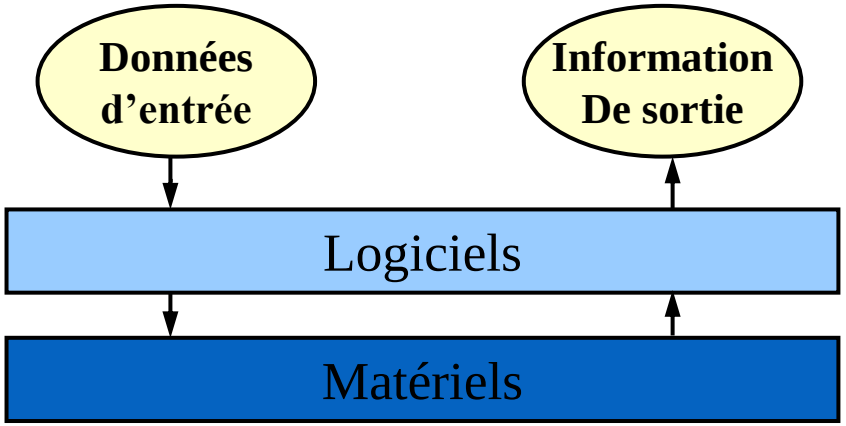
Science de **traitement automatique** et **rationnel** de l'**information.**

Un système informatique a pour buts :

- ☞ L'exécution réaliser des **tâches longues**, **répétitives** et **complexes**
- ☞ La production des **résultats précis en un temps court**

Système Informatique :

Ensemble des éléments **matériels** et **logiciels** permettant le traitement de l'information.



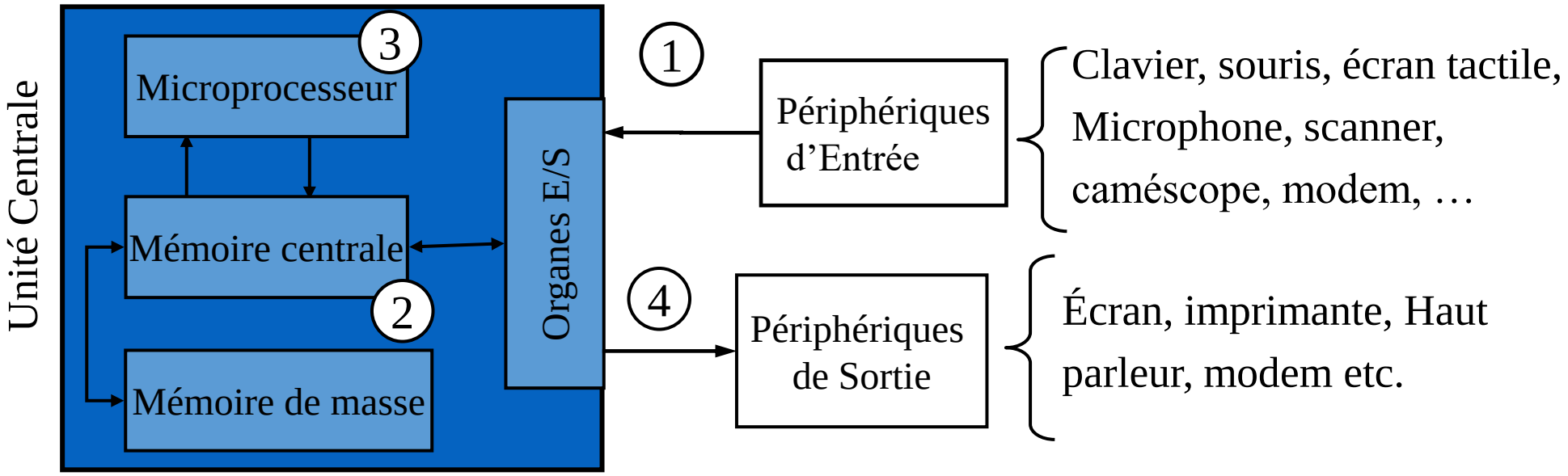
2. Fonctions d'un Systèmes Informatique

① La **collecte** des informations d'entrée **: Périphériques d'entrée**

② La **Mémorisation** (donnée + programmes) **: Mémoires**

③ Le **Traitement** des données **: microprocesseur**

④ La **restitution** des informations de sortie **: Périphériques de sortie**



Chap. I : Introduction à l'algorithmique

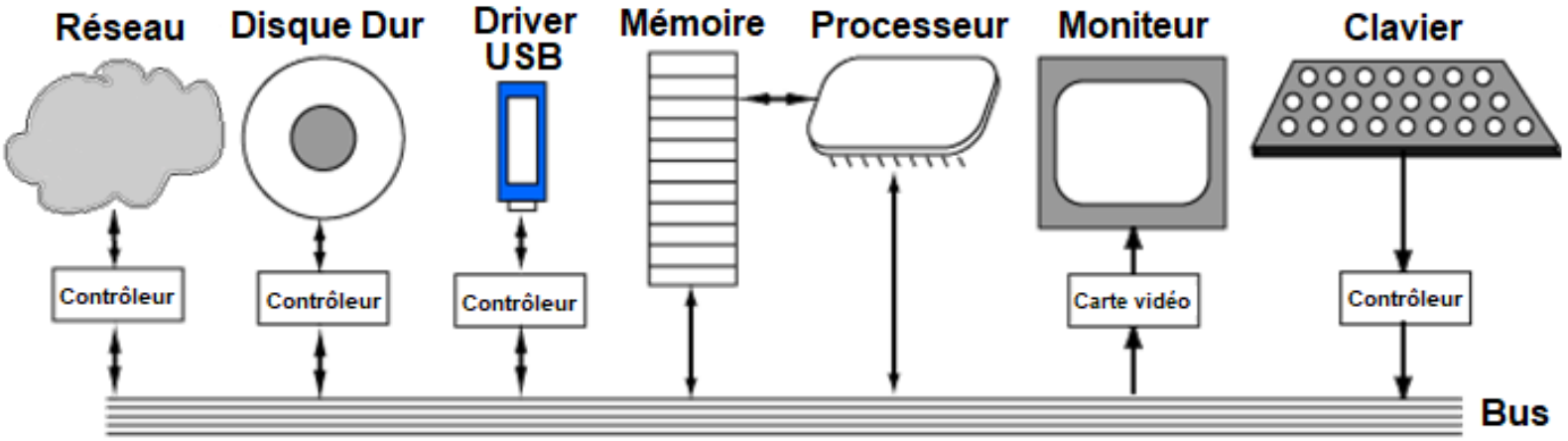
3. L'information

Elément de **connaissance** humaine susceptible d'être **représentée** à l'aide de conventions (**codages**) afin d'être **conservée**, **traitée** et **communiquée**.

Formes de l'information

Une information peut avoir les formes suivantes :

- **écrite** : lettres, factures, fiches techniques …
- **picturale** : dessins de voiture, schémas de bâtiment, graphiques, photographies, …
- **autres formes** : orale ou sonore, Tactiles, olfactives, …



4. Matériel

Ensemble de **circuits électroniques**. Chaque circuit est caractérisé par deux états : **état 1** (passage de courant) et **état 0** (Absence de courant). Les symboles **{ 0, 1}** constituent **l'alphabet** du **langage machine** (**binaire**) compréhensible par le matériel

Chap. I : Introduction à l'algorithmique

4.1. Les Mémoires

On distingue les classes de mémoires suivantes :

- La **mémoire centrale** :
  - **RAM** : contient les **données** et les programmes de façon **temporaire**
  - **ROM** : Contient le **BIOS** (programmes définis par le constructeur)
- La **mémoire de masse** (disquette, disque dur, CD/DVD ROM, USB …) :

  Stocke les données et les programmes de façon **permanente**

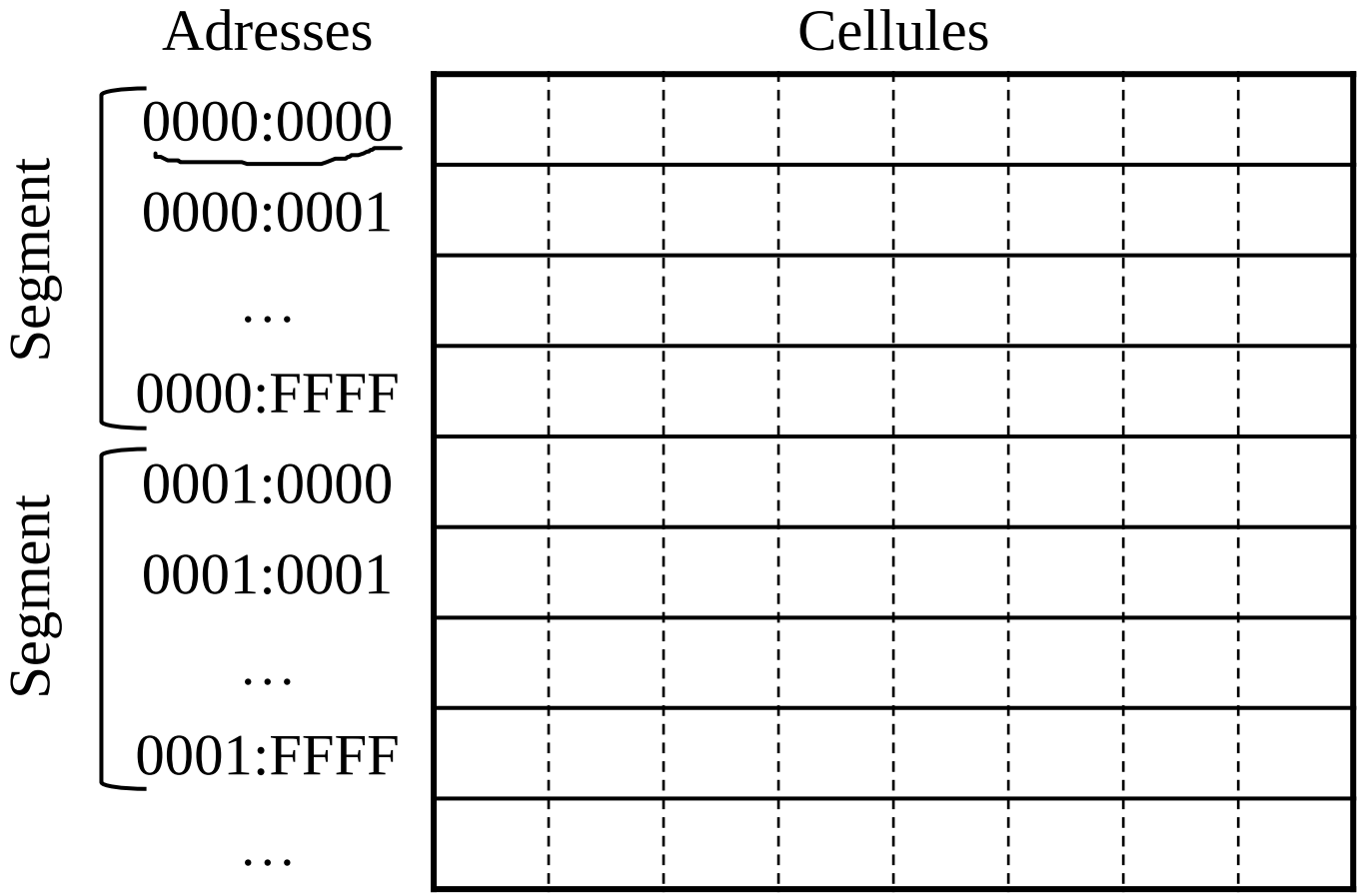
Unités de mémoires

- **Le bit** (Binary digIT) :Peut contenir un seul état binaire (0 ou 1)
- L'**Octet** (**O**) : Byte en anglais, représente un groupe de 8 bits
- **KiloOctet** (**KO**) : 1KO = 1024 O = 2^{10} O
- **MegaOctet MO**) : 1MO = 1024 KO = 2^{20} O
- **GigaOctet** (**GO**) : 1GO = 1024 MO = 2^{30} O
- **Tira Octet** (**TO**) : 1TO = 1024 GO = 2^{40} O

Schéma logique d'une mémoire

Une mémoire est vue logiquement comme une suite de **cellules** d'un octet chacune. Chaque cellule est caractérisée par :

- Une **adresse** : entier qui représente la position de la cellule
- Le **contenu** : suite de huit digits binaires

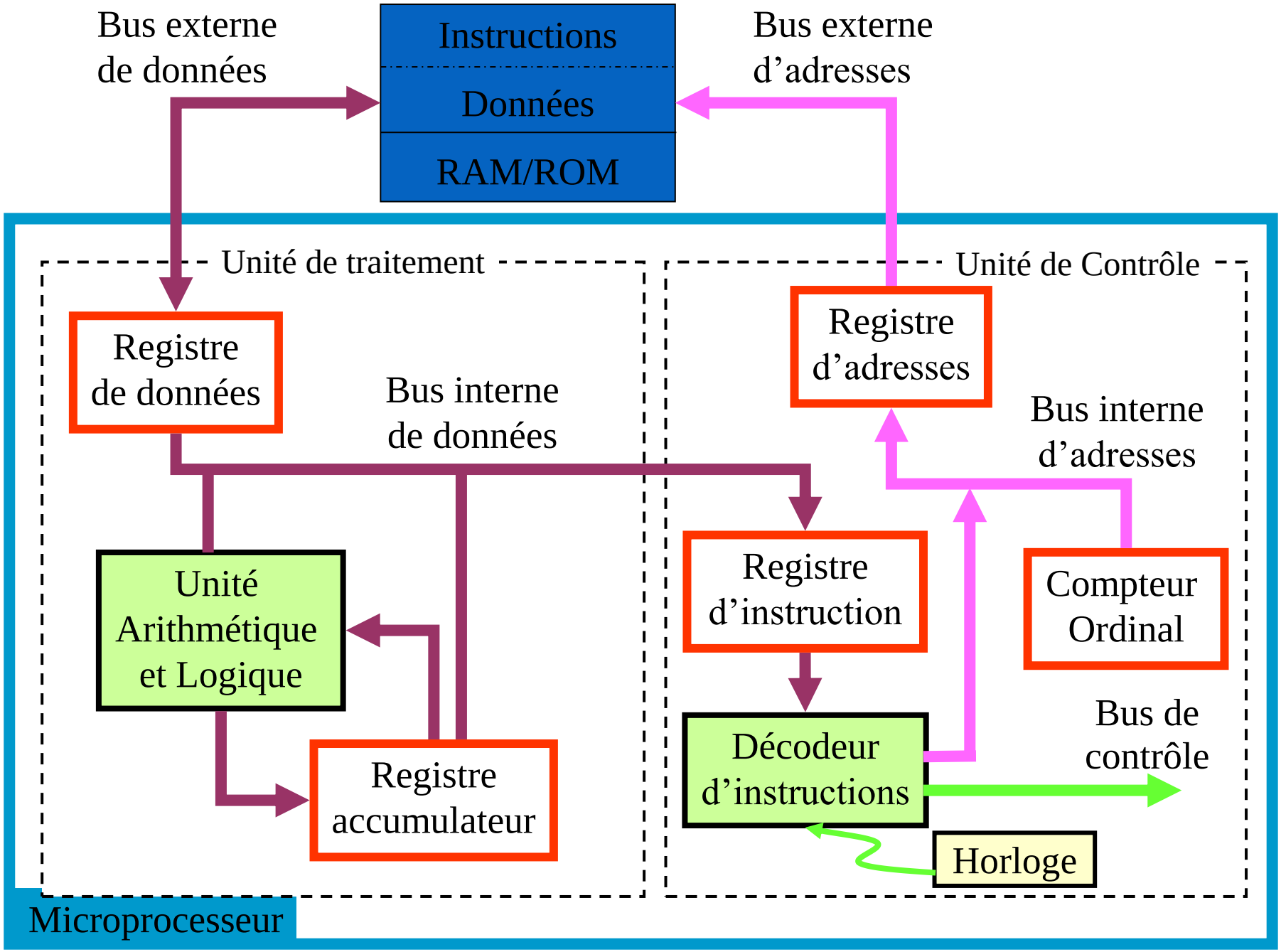


4.2. Le microprocesseur (CPU)

C'est le composant essentiel de l'ordinateur qui exécute les instructions (jeu d'instructions) et génère les commandes. Il est constitué essentiellement des éléments suivants :

- **L'unité arithmétique et logique** : effectue les calculs **arithmétiques** et **logiques**
- **L'unité de contrôle** : analyse les instructions à exécuter et génère les ordres de commande pour les autres composants
- **Horloge** : fixe la cadence d'exécution des instructions élémentaires. (GHz)
- **Registres** : mémoires très rapides de petite taille (8, 16, 32 , 64 , 128)bits
- **Mémoire cache** :.mémoire plus rapides que la RAM et sert à minimiser le nombre d'accès à la RAM
- **Bus** : permet le transfert des données (bus de données), d'adresses (bus d'adresses) et de commandes (bus de commandes)

Schéma interne du microprocesseur



5. Logiciel

Ensemble de **programmes** qui réalisent l'interface entre le matériel et l'utilisateur. Un Programme est une suite finie et ordonnée **d'actions** écrites dans un **langage** accepté par l'ordinateur

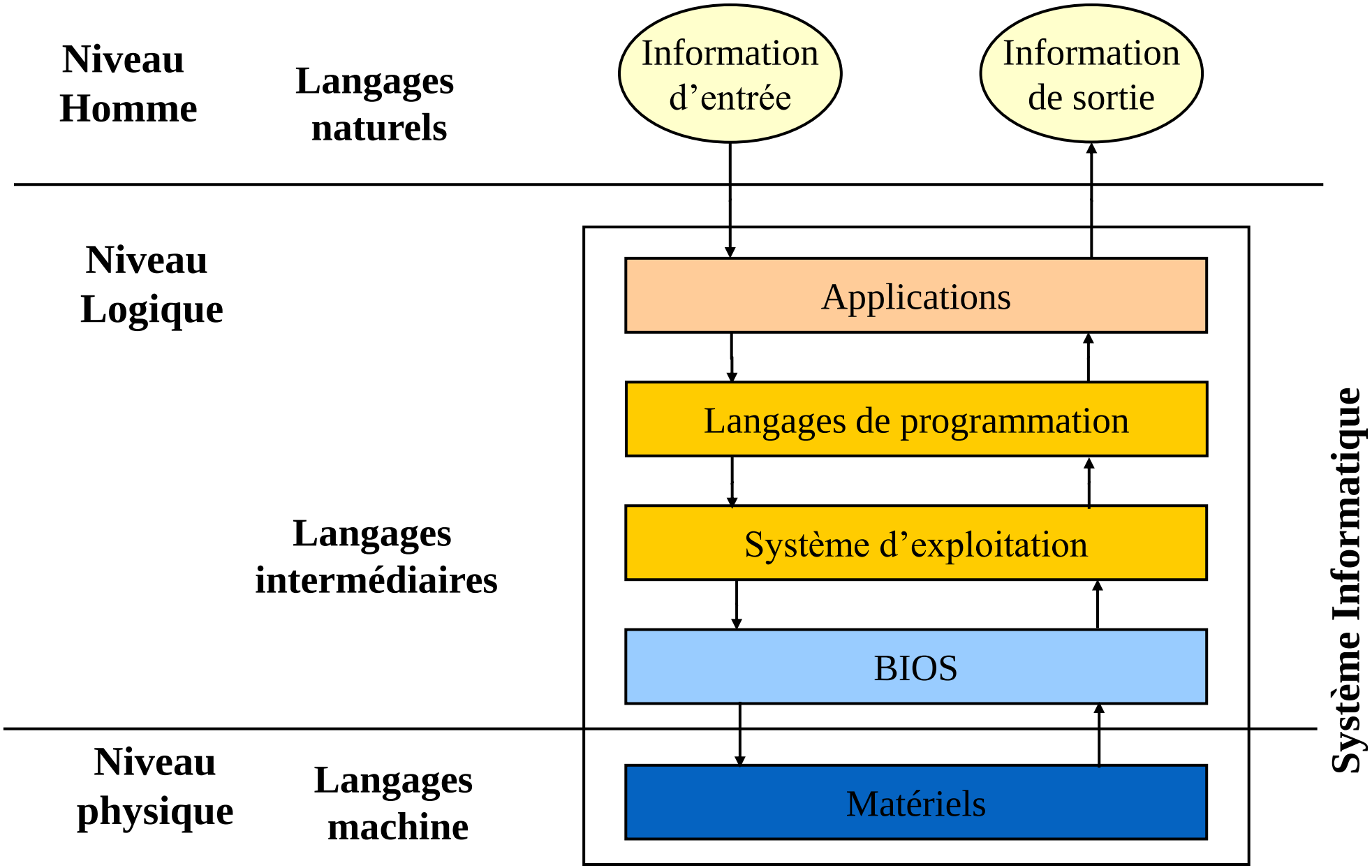
5.1. Types de Logiciels

On distingue quatre catégories de logiciels :

- **BIOS** : programme de base qui s'exécute au démarrage de l'ordinateur.
  - Identifie les différents composants et vérifie leur état
  - Permet de configurer certains périphériques
  - Offre des fonctions de bas niveau pour utiliser ces périphériques.
- **Systèmes d'exploitation** (Dos , Windows, Unix , Linux, MacOS,…) :
  - gère les ressources matérielles,
  - gère l'exécution des programmes,
  - gère le stockage des données dans les mémoire de masse,
  - gère les utilisateurs

- **Langages de programmation** : Environnements destinés pour produire des applications. On distingue 3 catégories: les compilateurs, les interpréteurs et les semi compilateurs
- **Applications utilisateurs** (programmes lancés par l'utilisateur) :
  - **Application standards** : traitements de texte , tableurs, PAO , …
  - **Applications spécifiques** : Logiciels de comptabilité, de dessin …

5.2. Couches d'abstraction d'un système informatique

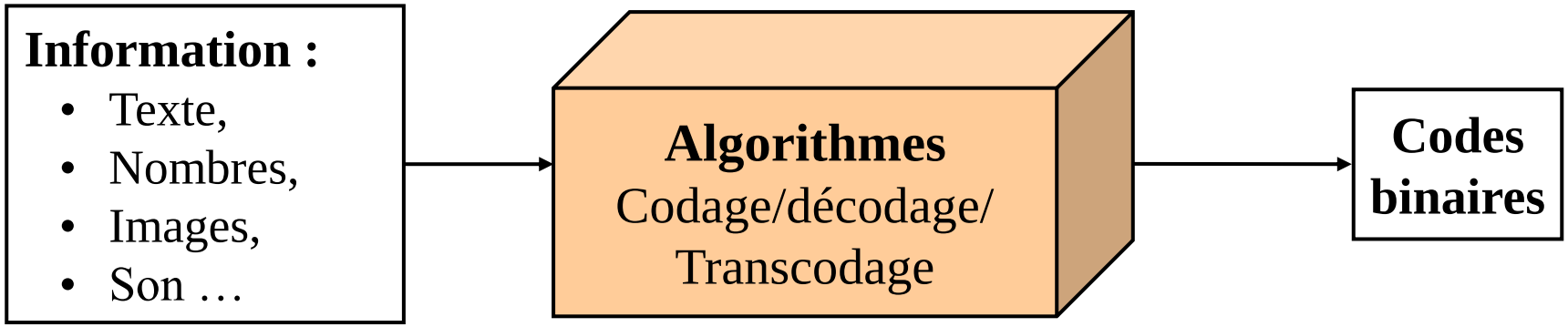


II. Notions de codage

1. Définitions

Dans un ordinateur, toutes les informations doivent être codées sous une forme physique représentée par une **suite de 0 et de 1 : langage binaire**. Il faut produire des **algorithmes** qui permettront à la machine de **coder les informations** à traiter : entiers, réels, caractères, images, sons ...

- Cette opération qui consiste à transformer l'information en code binaire unique correspondant s'appelle **codage**.
- Le **décodage** étant l'opération inverse qui permet de remonter à partir d'un code binaire à l'information correspondante.
- Le **transcodage** est le changement de code appliqué à des informations déjà codées.



Chap. I : Introduction à l'algorithmique

2. Systèmes de numération

Les systèmes de numérations **binaire**, **Octal** et **hexadécimal** sont très utilisés en informatique. Tout programmeur doit les connaître en plus du système **décimal**.

2.1. Principe d'une base

La **base** est le nombre qui sert à définir un **système de numération**. **exemples :**

Système de numération	Base	digits
Binaire	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Décimal	10	0,1,2,3,4,5,6,7,8,9
Hexadécimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

2.2. Conversion d'un nombre en base b quelconque en décimal

Quelque soit la base numérique b employée, elle suit la relation suivante :

$$(a_n a_{n-1} \ldots a_1 a_0)_b = (\sum_{i=0}^{i=n} a_i b^i)_{10} = (a_n b^n + \ldots + a_2 b^2 + a_1 b^1 + a_0 b^0)_{10}$$

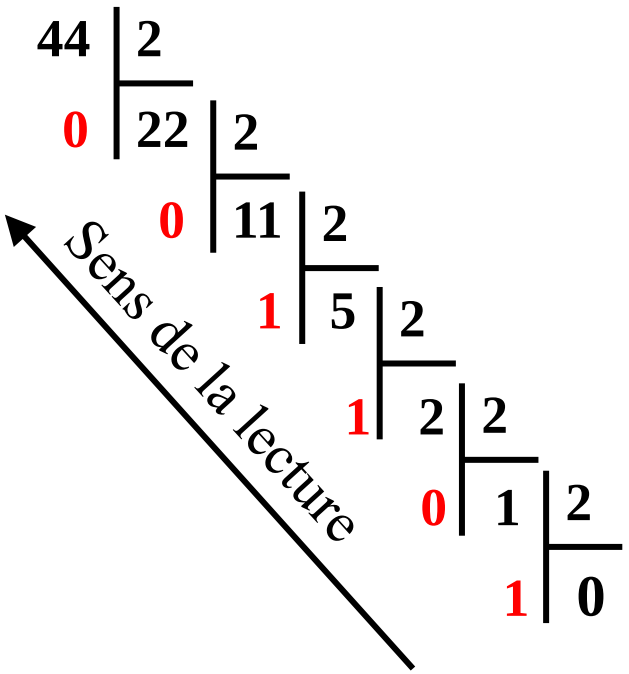
ou : a_i : digit de la base b de rang i ($0 \leq a_i < b$)

et : b^i : puissance de la base b d'exposant de rang i

Chap. I : Introduction à l'algorithmique

2.3. Décimal vers binaire

Pour obtenir l'expression binaire d'un nombre exprimé en décimal, il suffit de **diviser successivement ce nombre par 2** jusqu'à ce que le quotient obtenu soit égal à O. Les restes de ces divisions lus de bas en haut représentent le nombre binaire.

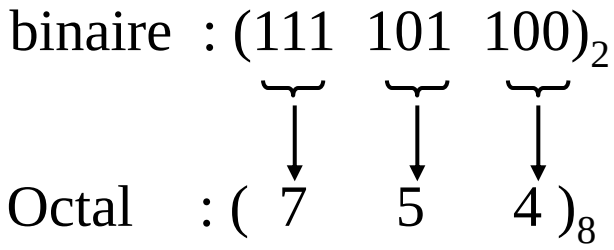


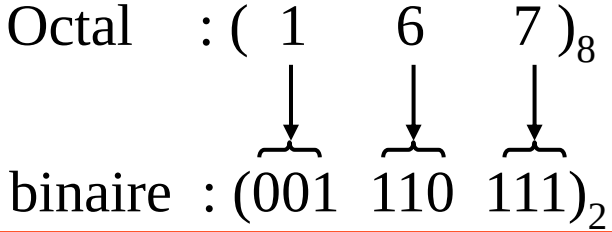
Exemple :

$(44)_{10} = (101100)_2$

2.4. Relation entre binaire et octal

La propriété d'équivalence entre chaque chiffre octal et chaque groupe de 3 chiffres binaires ($8=2^3$) permet de passer facilement d'un système à base 8 à un système à base 2 et vice versa.

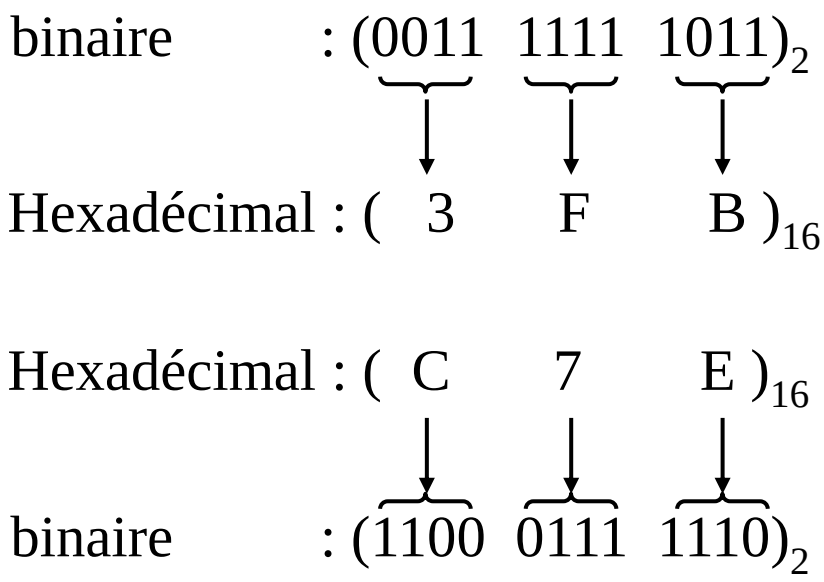




Exemples:

2.5. Relation entre binaire et hexadécimal

Le passage du binaire au hexadécimal et vice versa s'obtient par équivalence entre chaque chiffre hexadécimal et chaque groupe de 4 chiffres binaires ($16=2^4$).



Exemples:

3. Codage des entiers

deux modes de codage des entiers :

- **binaire pur**
- **complément à deux**

3.1. Binaire pur (entiers naturels)

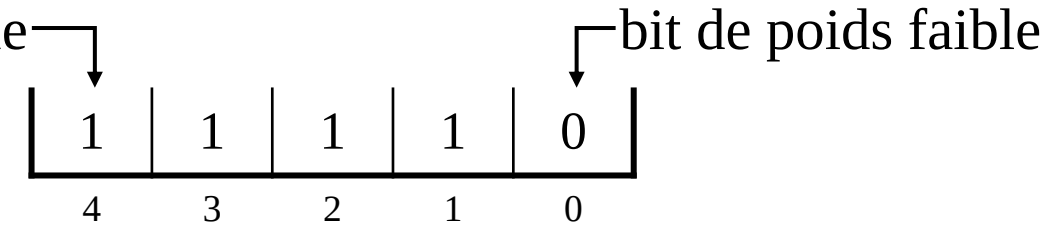
Pour coder un entier naturel n, il faut prévoir une mémoire minimale de p bits tel que : $\mathbf{2^{p-1} \leq n < 2^p}$

Inversement une mémoire de p bits est suffisante pour représenter le code binaire de tout entier $\mathbf{n \in [0 .. 2^p - 1]}$

Exemples:

Pour coder 30, le nombre minimal de bits nécessaire est 5 car :

$2^4 \leq 30 < 2^5$ $(16 \leq 30 < 32)$



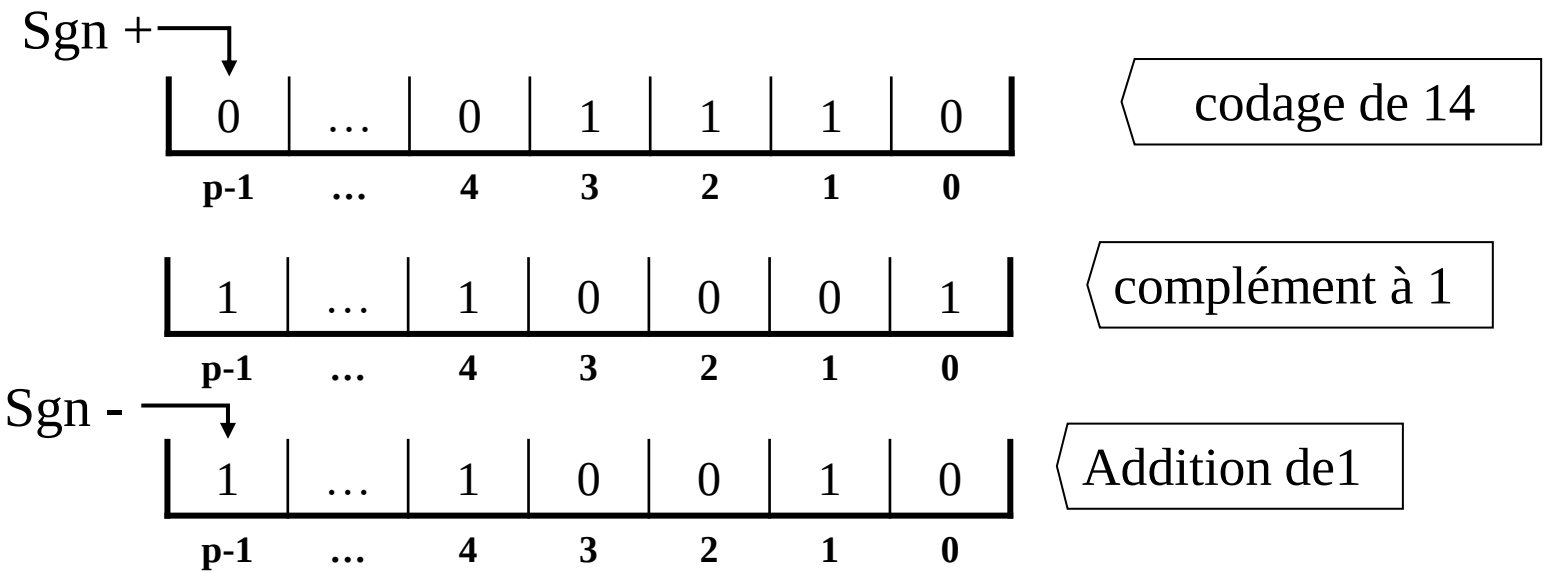
Une mémoire de 8 bits permet de coder tout entier de [0 .. 255]

3.2. Binaire Signé - complément à 2 (Entiers relatifs)

Ce codage permet de représenter en mémoire un entier relatif x par :

- le bit de poids fort représentant le signe (0 si x>0 et 1 si x<0)
- si x > 0, le codage en binaire naturel qui s'applique (le bit de signe vaut 0, les n bits restants codent le nombre),
- si x < 0, alors
  - on code |x| en binaire.
  - on complémente tous les bits de la mémoire.
  - on additionne 1 au nombre binaire de la mémoire (addition binaire).

Exemple :représentation binaire du nombre -14 sur p bits :



Une mémoire de p bits est suffisante pour représenter le code binaire de tout entier relatif $\mathbf{n \in [-2^{p-1} .. 2^{p-1}-1]}$

Par exemple une mémoire de 8 bits pour coder tout entier de l'intervalle [-128 … 127]

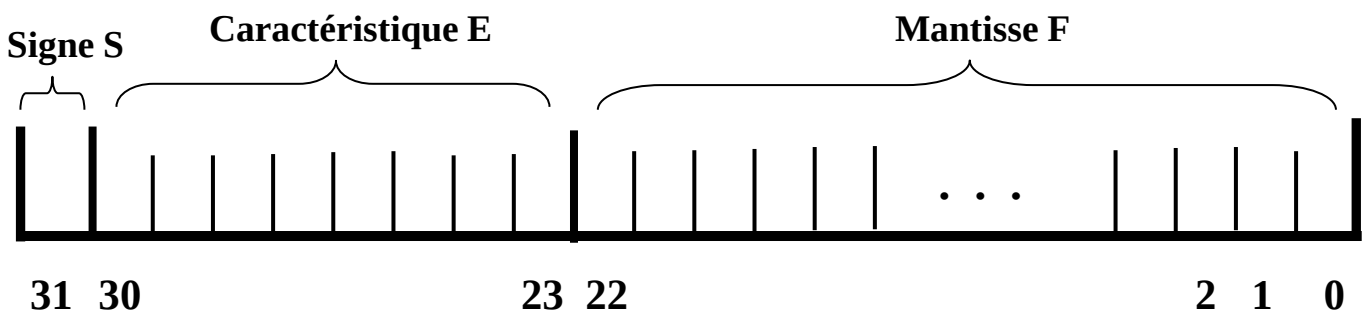
4. Codage des nombres Réels

Un réel x peut être représenté dans la base 2 selon un standard défini par l'IEEE :

$$\mathbf{x} = (-1)^{s} \; 2^{E-127} \; 1.\mathbf{F}$$

signe	+ -	**s**
exposant	caractéristique	**E**
partie décimale	mantisse	**F**

En général on représente un réel sur 4 octets (32 bits) par :



Décomposition d'un nombre réel décimal en binaire

Soit le nombre 0.8 à convertir en binaire. Le code correspondant est :

S = 0	E=126	F
0	01111110	10011001100110011001100

On constate d'abord que son signe est positif, donc S=0. On cherche ensuite à le décomposer en une somme de puissances de 2.

$0.8 = 2^{0} \times 0.8$

$0.8 \times 2 = 1.6$ donc $0.8 = 2^{-1} \times 1.6 = 1 \times 2^{-1} + 0.6 \times 2^{-1}$

$0.6 \times 2 = 1.2$ donc $0.8 = 1 \times 2^{-1} + 1.2 \times 2^{-2}$

$0.2 \times 2 = 0.4$ donc $0.8 = 1 \times 2^{-1} + 1 \times 2^{-2} + 0.4 \times 2^{-3}$

$0.4 \times 2 = 0.8$ donc $0.8 = 2^{-1} \times 1 + 2^{-2} \times 1 + 2^{-3} \times 0 + 2^{-4} \times 0.8$

$0.8 \times 2 = 1.6$ donc on retrouve une expression déjà rencontrée qui se répéter infiniment

$0.8 = (-1)^0 \times 1.1001100110011001.... \times 2^{-1}$

5. Codage des caractères.

Les caractères sont représentés sur **8** bits. La convention de **codage ASCII** est la plus fréquente. Elle fait correspondre à chaque caractère de l 'ensemble des caractères une valeur entière de 0 à 255.

On distingue deux catégories de caractères :

- Caractères **non imprimables** : correspondent à des caractères de contrôle. Ils ont des codes compris entre 0 et 32
- Caractères **imprimables** : Lettres majuscules (65 à 92), Lettres minuscules (97 à 124),les chiffres (48,57) et les symboles

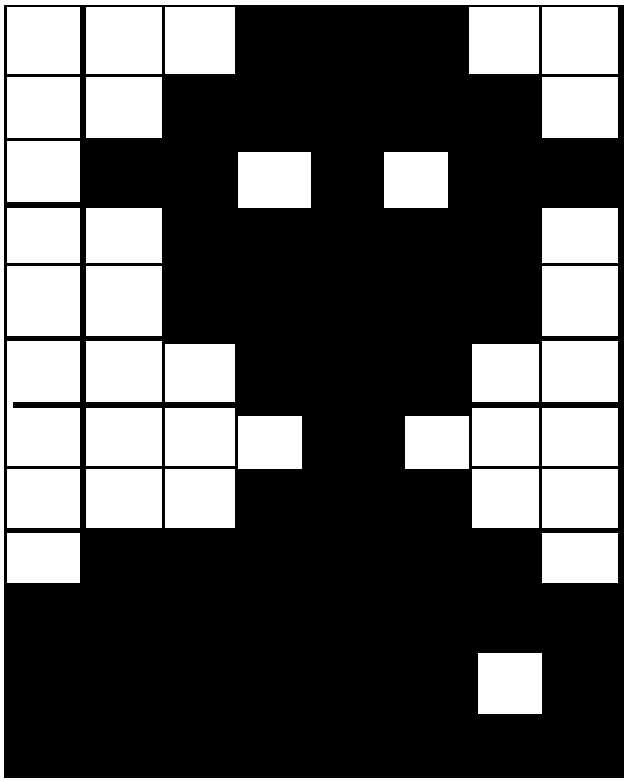
6. Codage des images

Le codage des images consiste à superposer une grille très fine à l 'image. Chaque carré de la grille appelé pixel est assimilé à une seule couleur coloré. En fonction du nombre de couleurs, chaque couleur est représenté par un code.

Pour une image monochrome le NOIR est codé 1, et le BLANC est codé 0. L 'image est donc codée sous forme d'un tableau à deux dimensions de 0 et de 1.

Exemple

Image 12 lignes de 8 colonnes



Codage binaire

00011100
00111110
01101011
00111110
00111110
00011100
00001000
00011100
01111110
11111111
11111101
11111111

Codages des couleurs en RGB

Une couleur codée en **RGB** se présente comme un nombre hexadécimal à six chiffres : **FF06C3** par exemple. Chaque paire de chiffres est dédiée à une couleur primaire. Sur le même exemple, cela donne : FF pour le **rouge**, 06 pour le **vert** et C3 pour le **bleu** : soit les valeurs décimales 255, 6 et 211.

Chacune de ces valeurs indique **l'intensité** avec laquelle la lampe correspondant va être allumée. Soit : 100 % pour le rouge, 2.35 % pour le vert et 82.75 % pour le bleu.

Toutes les lampes au maximum de leur intensité **FFFFFF** fournissent du **blanc**, alors que toutes les lampes éteintes **000000** font du **noir**.

6. Codage des sons

Le codage d'un son passe par sa **transformation** en un **signal électrique** continu puis son **échantillonnement** ce signal électrique et le **numériser**.

III. Notions d'algorithmique

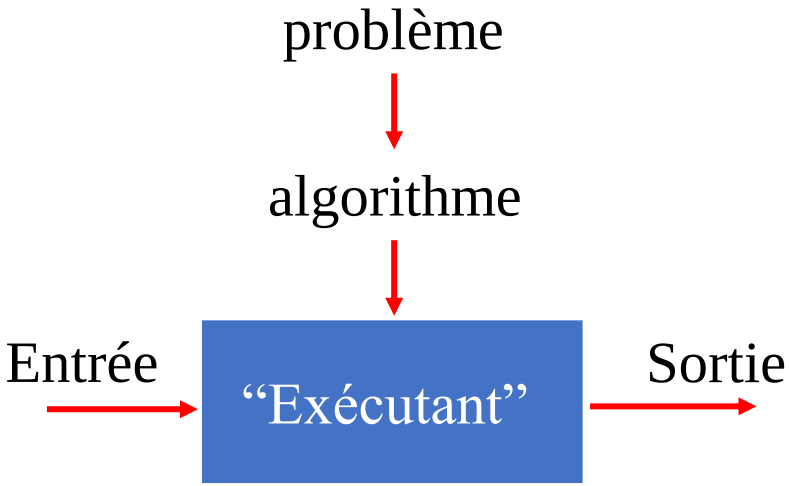
1. Définitions

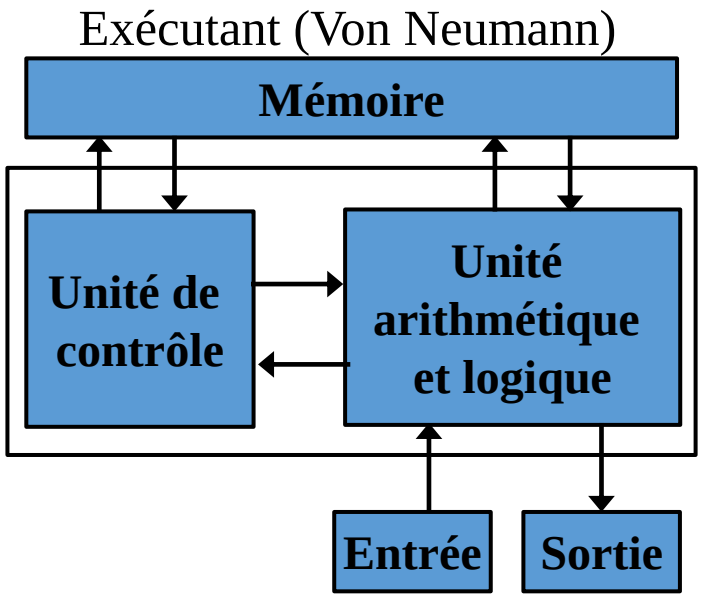
Algorithme

Un algorithme est une **séquence finie d'instructions** non ambiguës énoncé dans un **langage formel** pour résoudre un **problème**, c'est-à-dire pour obtenir, en un **temps fini**, une **sortie** requise pour toute **entrée valide**.

Algorithmique

L'algorithmique s'intéresse à **l'art de construire des algorithmes** ainsi qu'à caractériser leur **validité**, leur **robustesse**, leur **réutilisabilité**, leur **complexité** ou leur **efficacité**.





Chap. I : Introduction à l'algorithmique

Programme

Une séquence d'instructions destinées à être **exécutées par un ordinateur**.

Code source : programme exprimé dans un **langage de programmation** compréhensible par un être humain, et ne pouvant pas être exécuté directement par l'ordinateur.

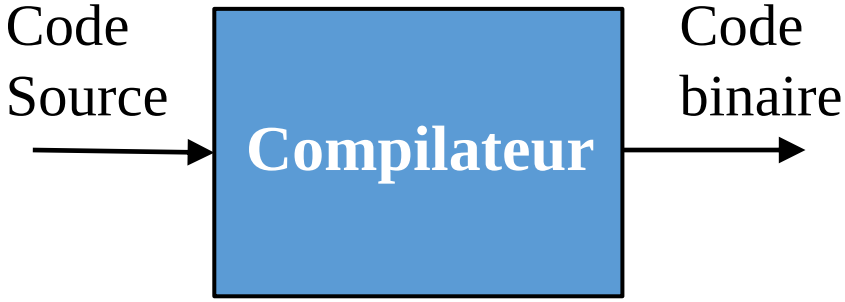
Code binaire : programme exprimé en langage machine, pouvant être directement exécuté par l'ordinateur

Compilateur

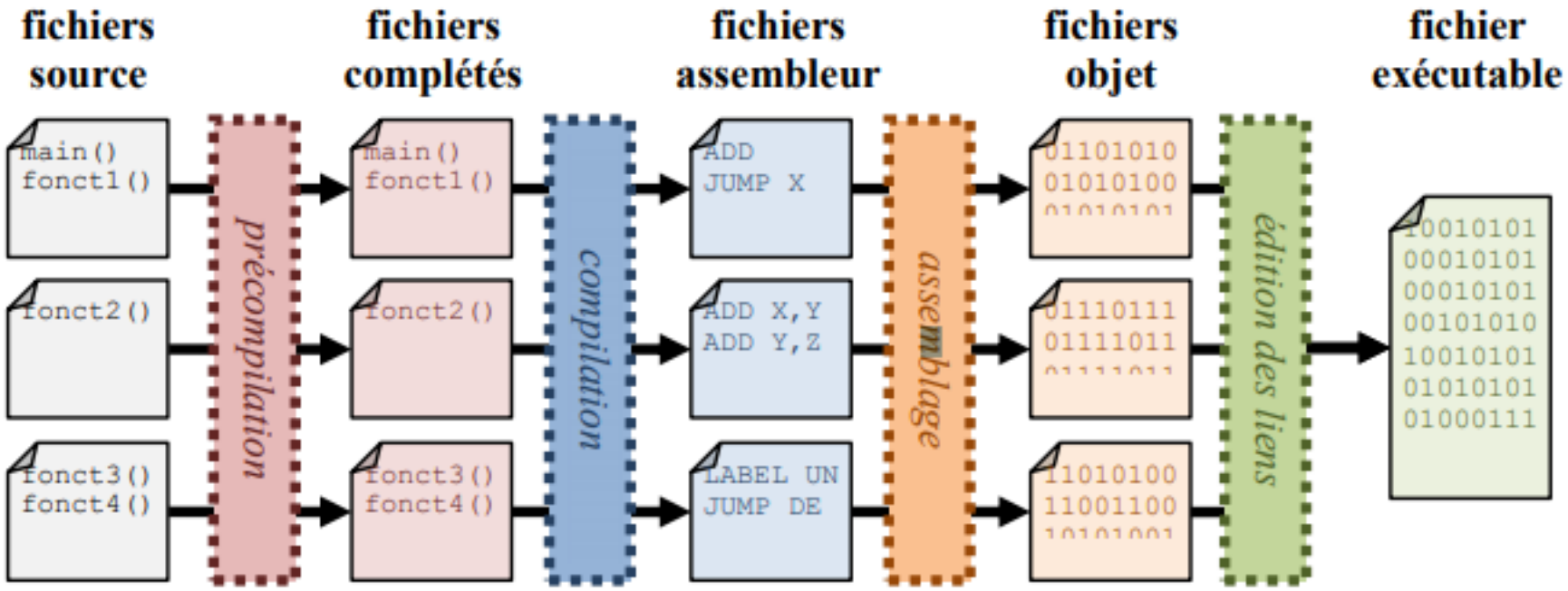
Programme système qui permet de de **transformer** un **code source** en **code binaire**, de manière à obtenir un **fichier exécutable** à partir d'un **fichier source**

Exemple de langages compilés

C, C++, Fortran, …



Etapes de compilation en langage C



Interpréteur : permet de **traduire** et **d'exécuter chaque instruction** du programme source (Lisp, Prolog, …)

Semi-compilateur : Basé sur un mécanisme qui combine les deux techniques (compilation pour une machine virtuelle et interprétation pour la machine physique)

Exemples : Java, Python, C#, …

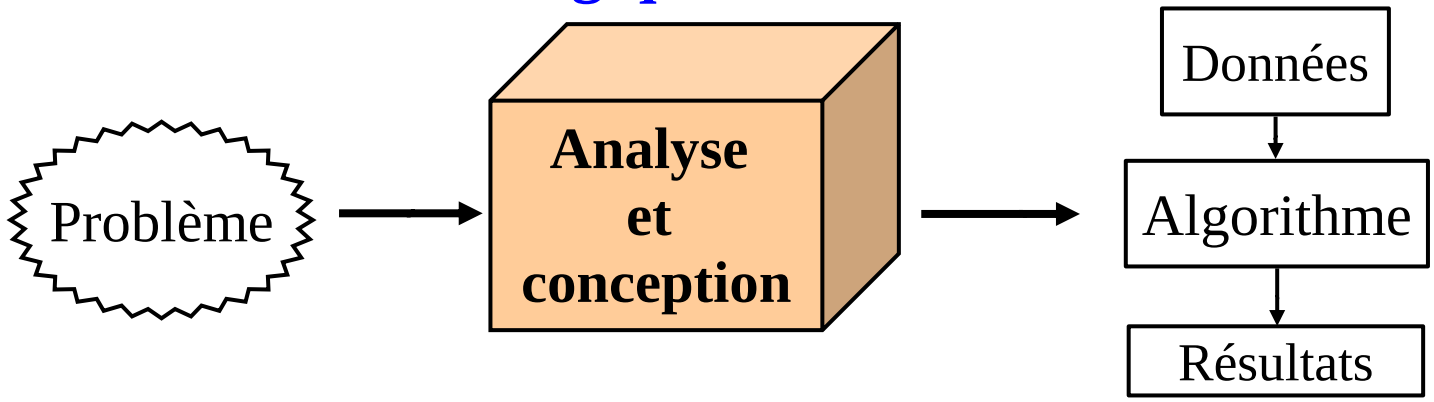
2. Caractéristiques d'un algorithme

Un bon algorithme doit être :

- **Fini** : Le nombre d'étapes de l'algorithme doit être fini
- **Précis** : Toute étape doit être définie parfaitement (sans ambiguïté).
- **Exécutable :** Chaque étape de l'algorithme doit être exécutable par l'exécutant
- **Efficace** : Produit les résultat en un minimum de ressources et de temps
- **Lisible** : doit être compréhensible même par des non informaticiens (formulé dans un langage formel indépendamment des détails techniques)
- **Réutilisable** : Structuré en entités réutilisables (entités fonctionnelles, Type abstrait de données, Modules, …)

Remarque : A un problème, il peut exister plusieurs algorithmes. Il faut trouver **le meilleur algorithme** pour le résoudre, le plus **efficace**.

3. Démarche méthodologique



3.1. Besoin de méthode

Pour concevoir une application on doit recourir à une **méthode** pour bien aborder :

- Complexité de la **structure de données**
- Complexité des **traitements à effectuer** …

3.2. Composantes d'une méthode de développement

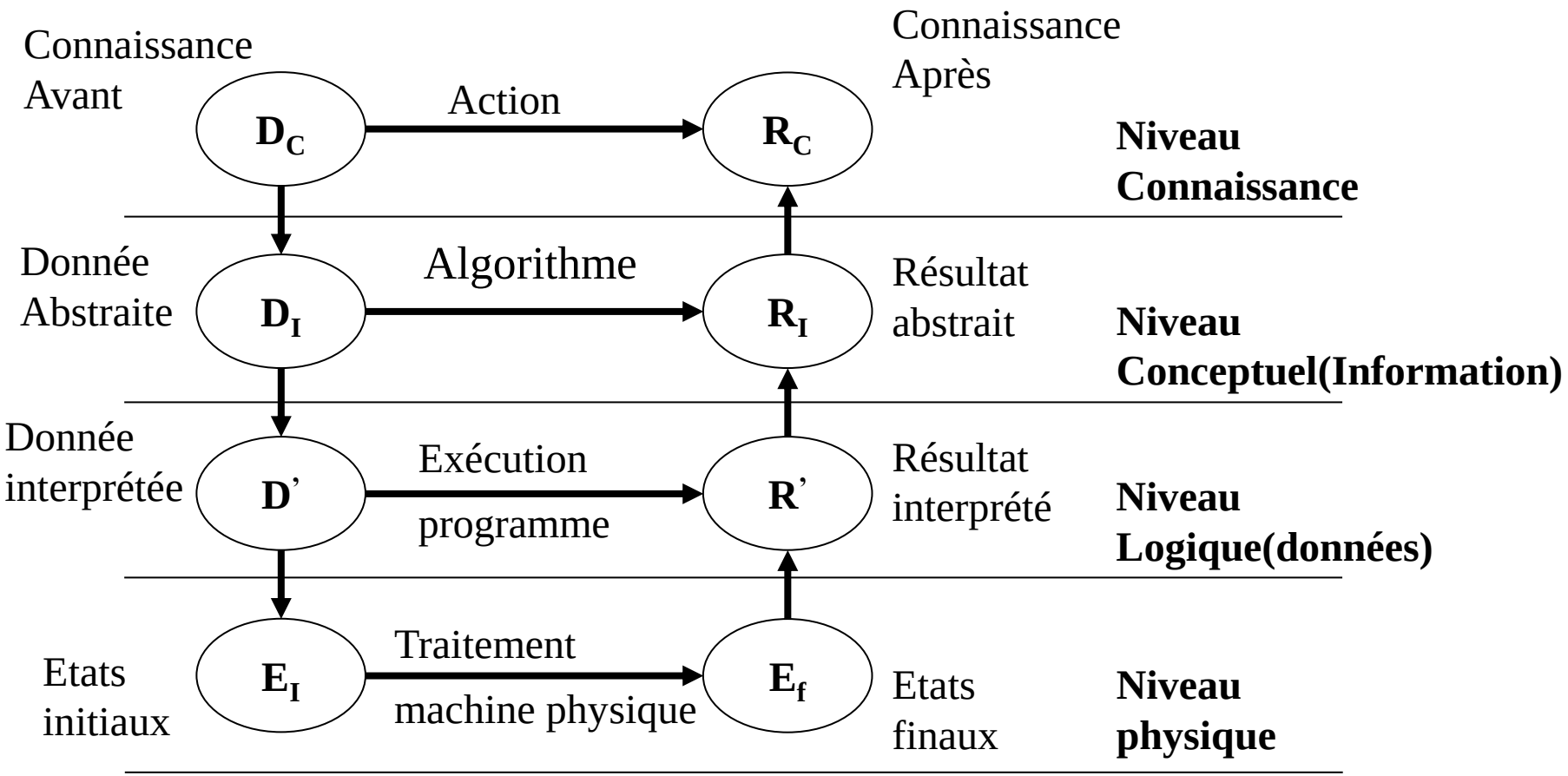
Une méthode de développement est composée des éléments suivants :

- Des **principes** : **abstraction** par niveaux, **diviser/réduire/transformer** pour **régner**
- Une **démarche** : Étapes de mise en œuvre
- Des **outils** : Des **Langages** , Des **modèles**

a. abstraction

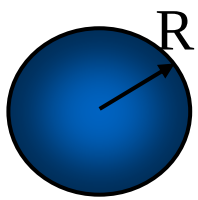
Le développement d'une application consiste à représenter les éléments de connaissances à des **niveaux d'abstraction différents** pour descendre du niveau **humain** au niveau de la **machine**

On distingue les quatre niveaux d'abstraction suivants :



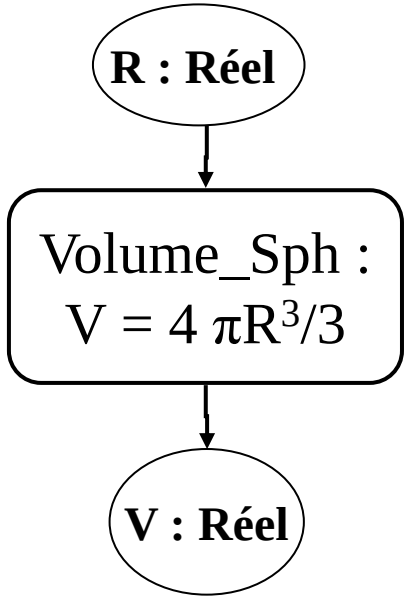
Abstraction - Exemple

Calcul du volume d'une sphère



$V = 4 \pi R^3/3$

Niveau Connaissance




```
Algorithme Vol_Sph
    Const    Reel Pi = 3.14
    Reel     R, V
Début
    Lire (R )
    V ← 4 * Pi * R* R *R /3
    Ecrire ("Volume = ",V)
Fin
```

Niveau information

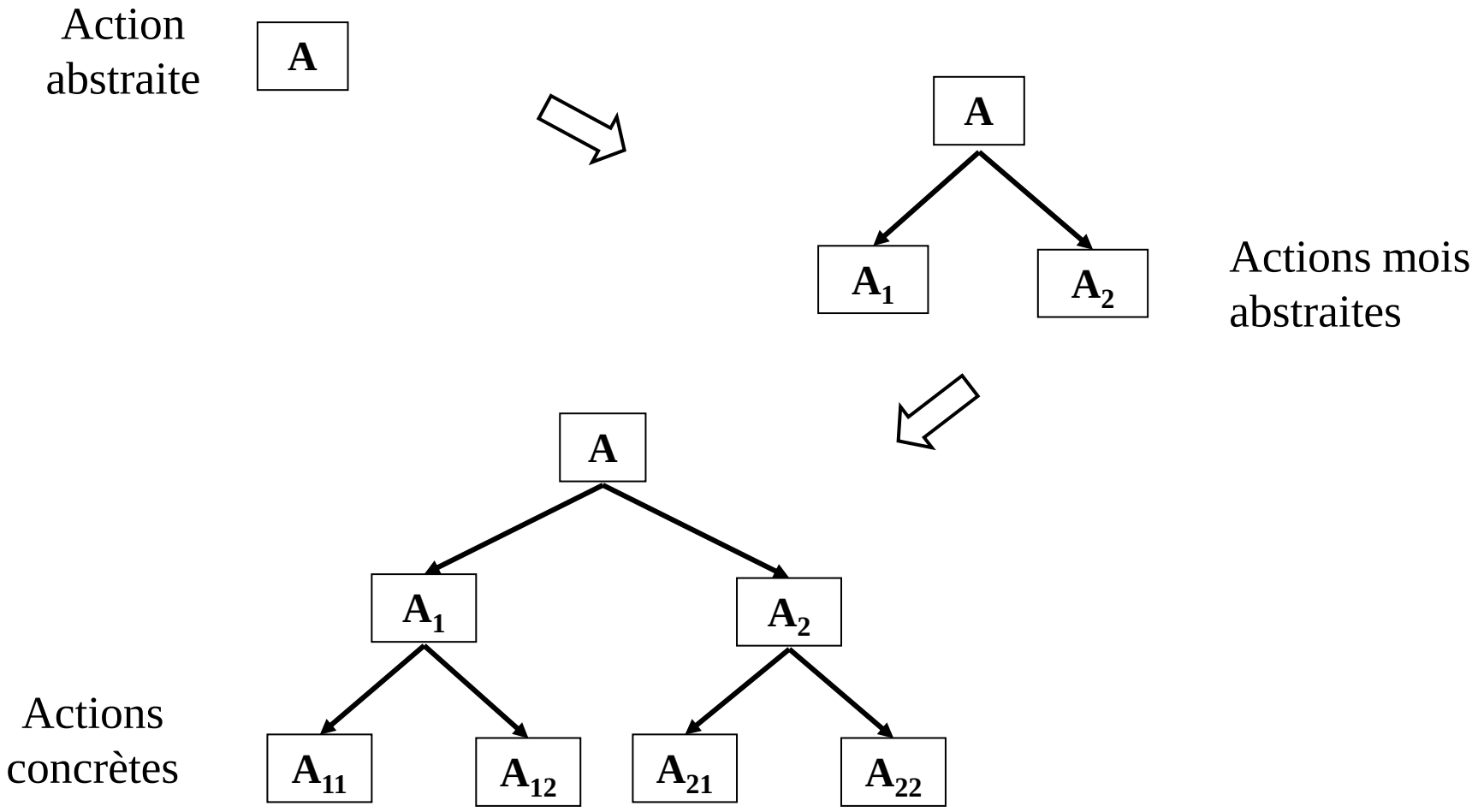
```
#include<stdio.h>
#define cube(x)   (x)*(x)*(x)

const  float Pi = 3.14 ;
float    R, V;
```

```
int  main(){
    scanf("%f" ,&R );
    V = 4 * Pi * cube(R) /3 ;
    printf(" volume : %.3f\n",V);
    return 0;
}
```

Niveau Logique

b. Diviser pour régner : Analyse descendante

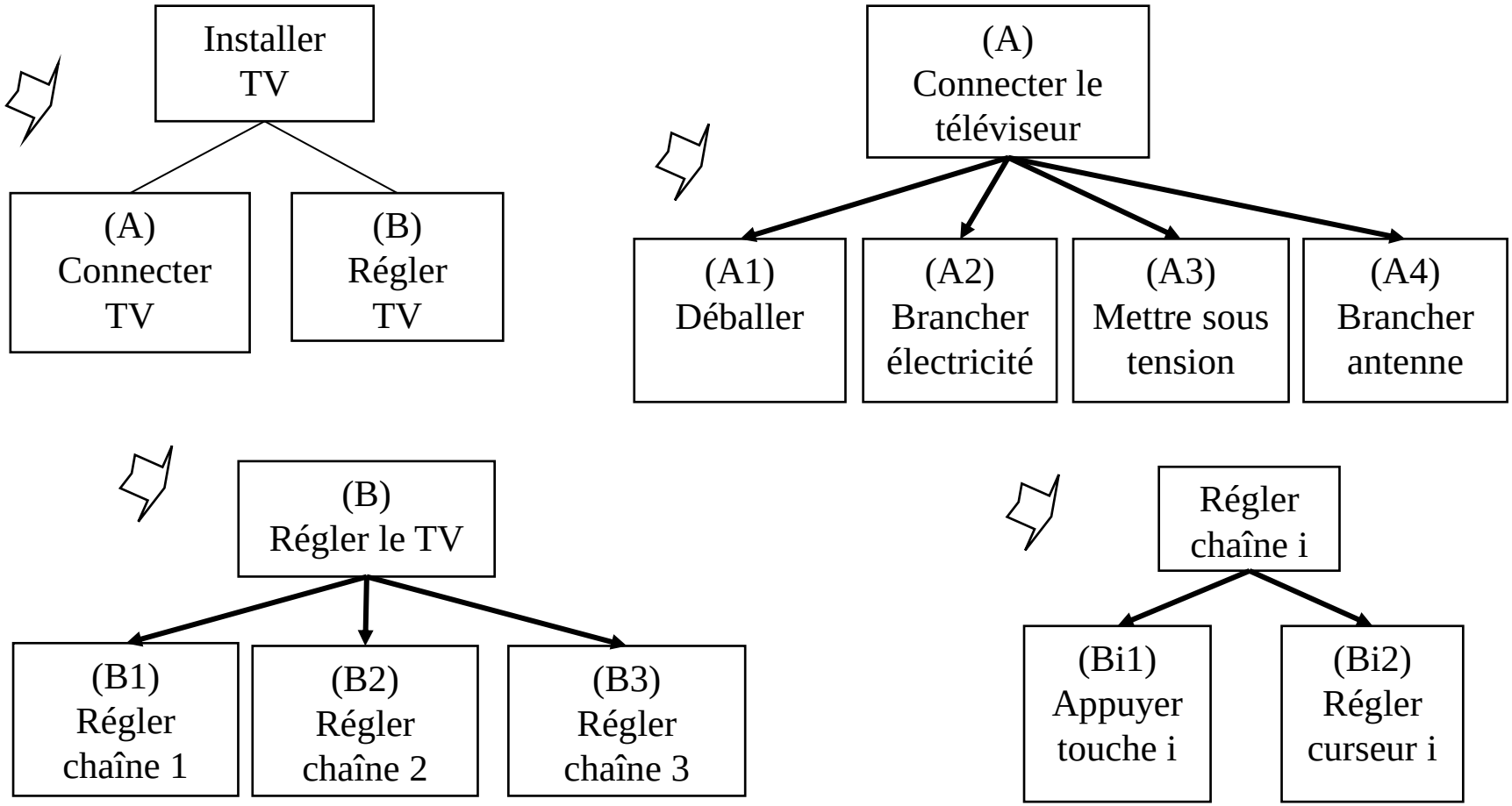
Diviser pour régner consiste à **décomposer** le problème complexe à résoudre en plusieurs **sous problèmes** moins complexes. A refaire cette décomposition sur les sous problèmes jusqu'à obtenir des sous problèmes faciles à résoudre.



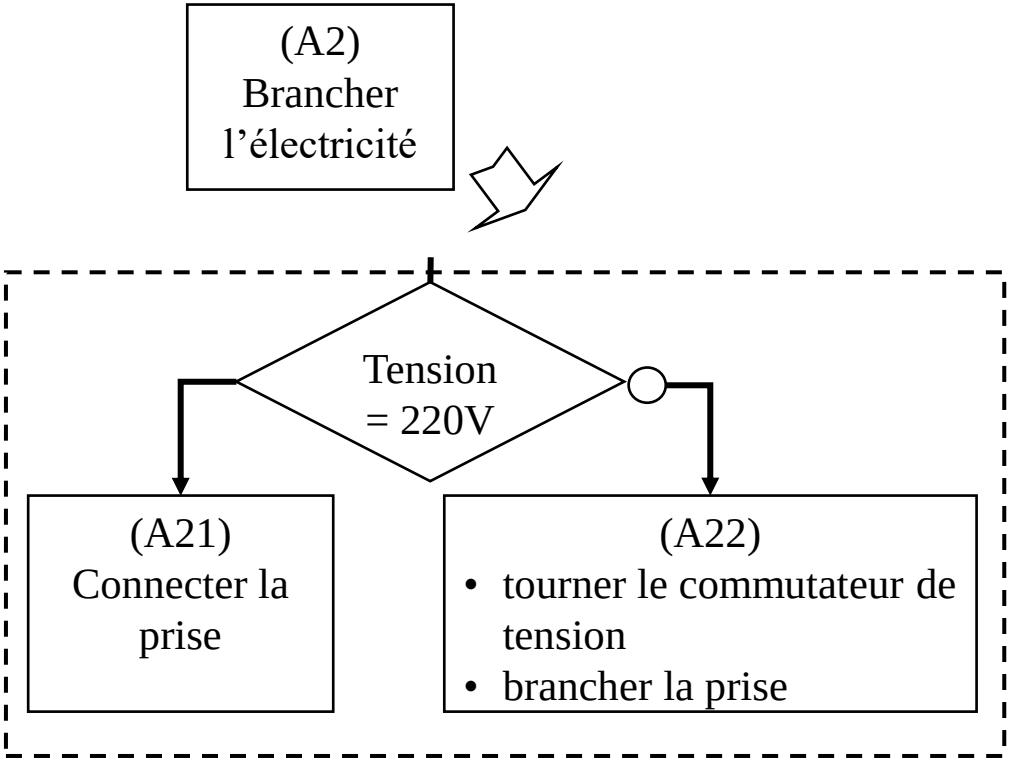
Exemple1 : Installation d'un téléviseur

Installer TV

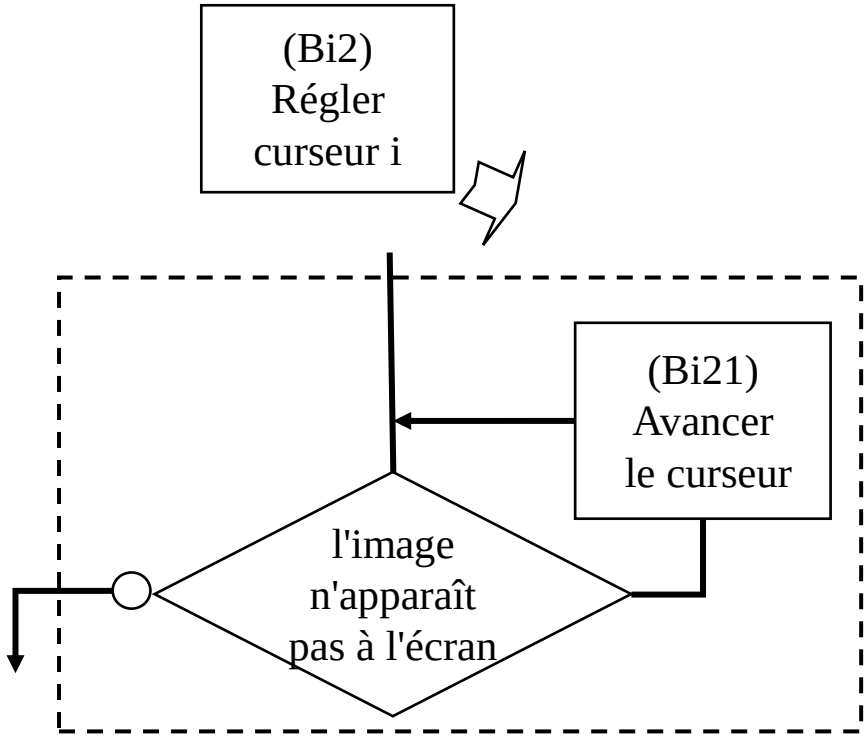
Enoncés séquentiels



Enoncé Conditionnel



Enoncé répétitif



Exemple 2 :

Déterminer la liste des 6 premiers nombres **d'Armstrong**.

Soit x un nombre entier qui s'écrit dans la base 10 sous la forme $x = d_{n-1}\ldots d_1 d_0$

$d_i \in \{0,1, \ldots, 9\}$ étant le digit numéro i.

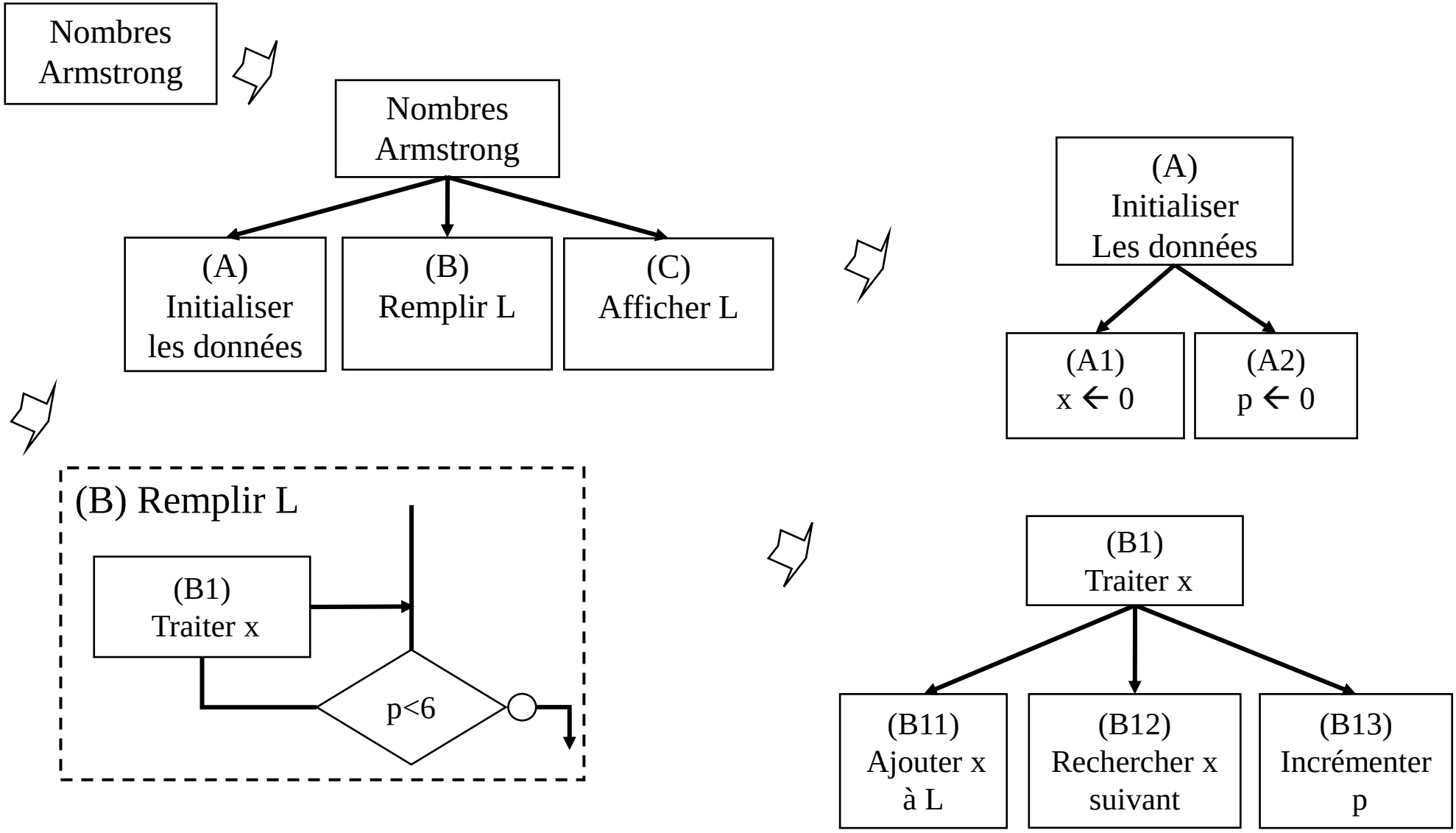
x est un nombre d'Armstrong ssi :

$$x = \sum_{i=0}^{n-1} d_i^3$$

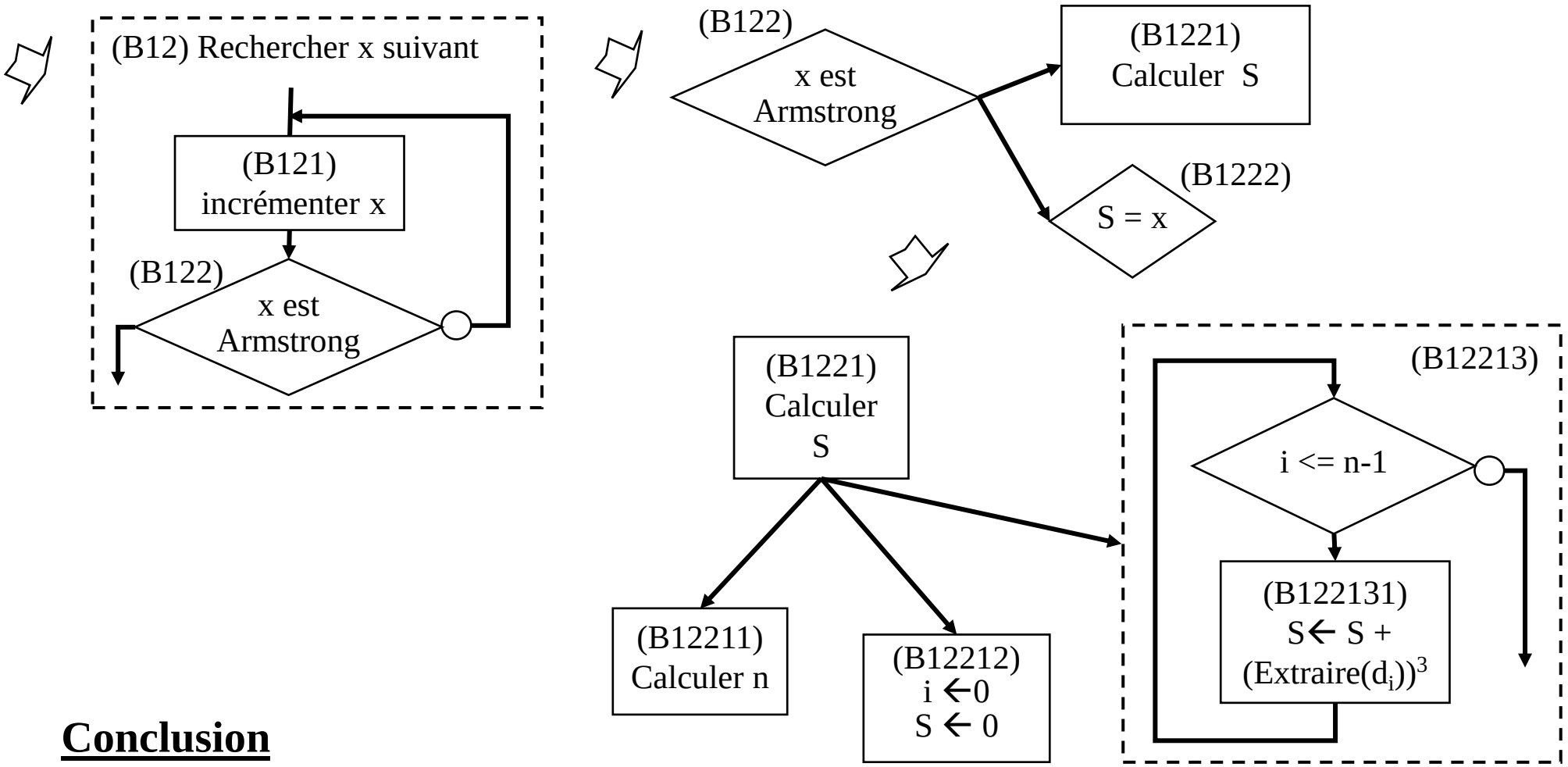
Chap. I : Introduction à l'algorithmique

Les données :

- x : un nombre d'Armstrong
- p : nombre de nombres d'Armstrong trouvés
- n : nombre de chiffres de x
- d_i : digit n° i du nombre x
- S : Somme des cubes de d_i
- L : Liste des nombres d'Armstrong



Chap. I : Introduction à l'algorithmique



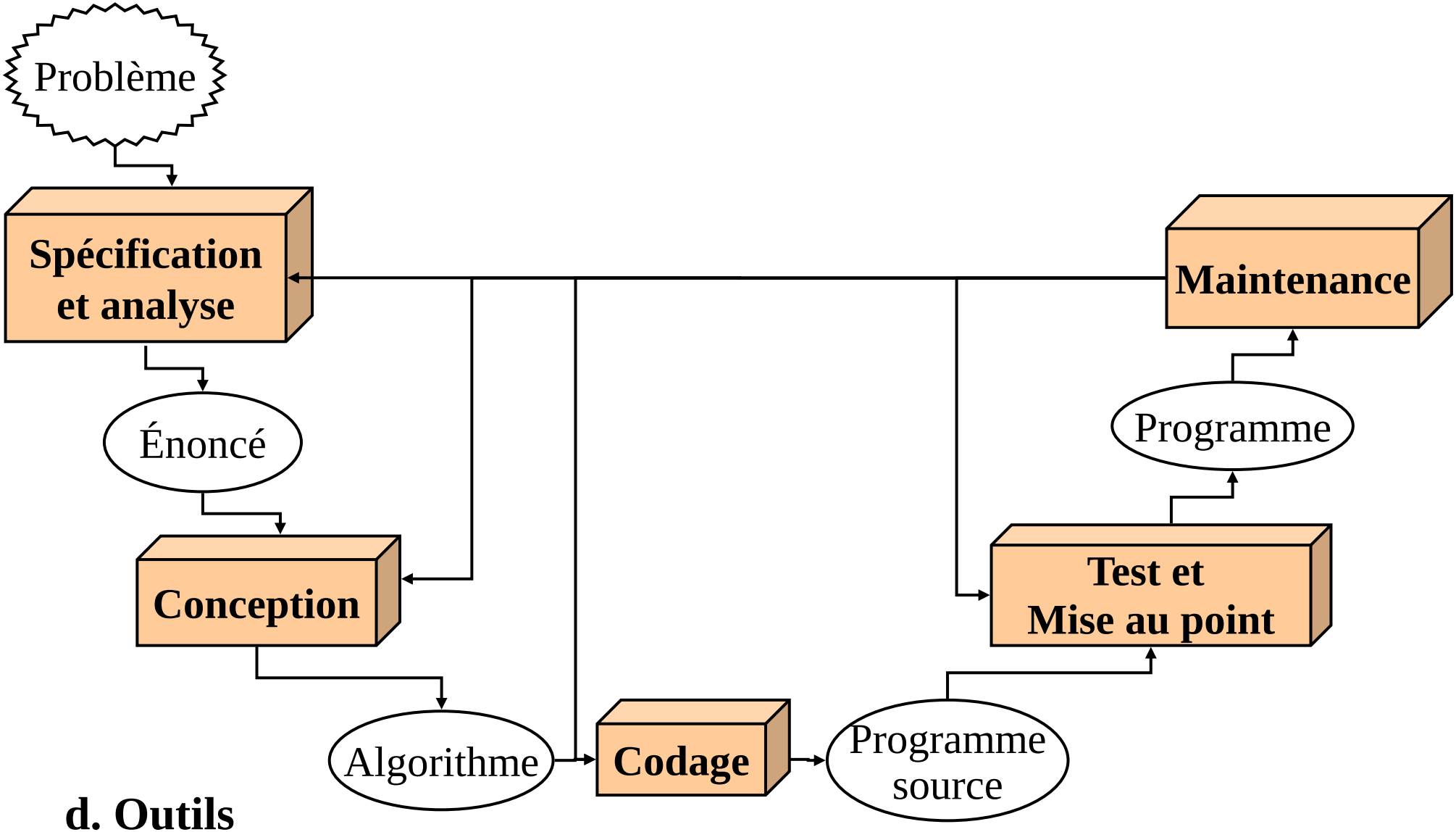
Conclusion

La solution à un problème bien défini peut être formulée comme une suite des trois énoncés suivants :

- **Séquentiel** : suite d'étapes
- **Conditionnel** : choix entre deux étapes suivant une condition
- **Répétitif** (itératif) : répétition d'une étape

c. Démarche

Le développement d'une application passe par les différentes phases suivantes :



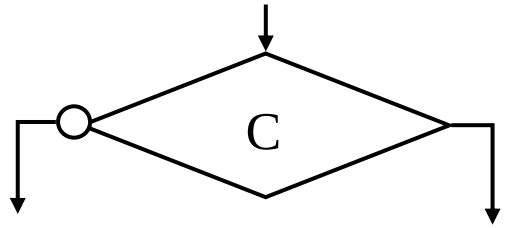
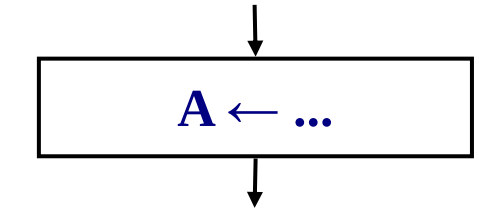
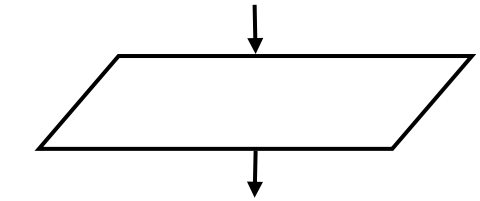
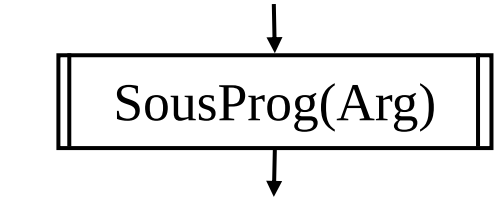
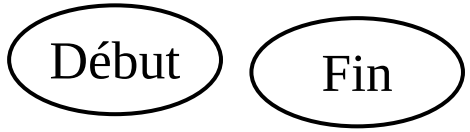
d. Outils

Langages : Outil textuel pour écrire des algorithmes (pseudo code = français)

Chap. I : Introduction à l'algorithmique

Organigramme

Représentation schématique utilisant des **symboles graphiques** pour décrire les étapes de résolution d'un problème :

	Marque le passage d 'une action à une autre
	Rupture de séquence marquant un choix conditionnel d 'exécuter une action suivant la valeur de la condition C
	Action de calcul pour modifier la valeur d'une variable
	Action d 'entrée(lecture d 'une donnée)/Sortie (affichage d 'un résultat)
	Action d'appel à un sous programme
	Marque le début et la fin de traitement

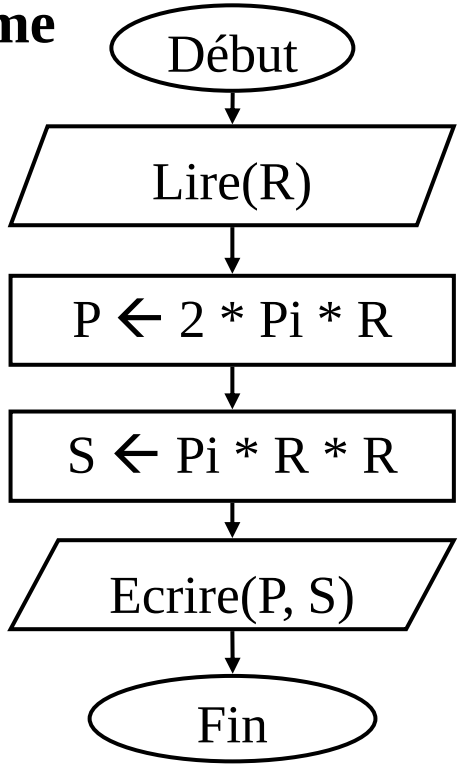
Chap. I : Introduction à l'algorithmique

4. Exemples d'algorithmes

Algorithme1 : Calcul du **périmètre** P et de la **surface** S d 'un **disque** de **rayon** R .

Spécification

Organigramme



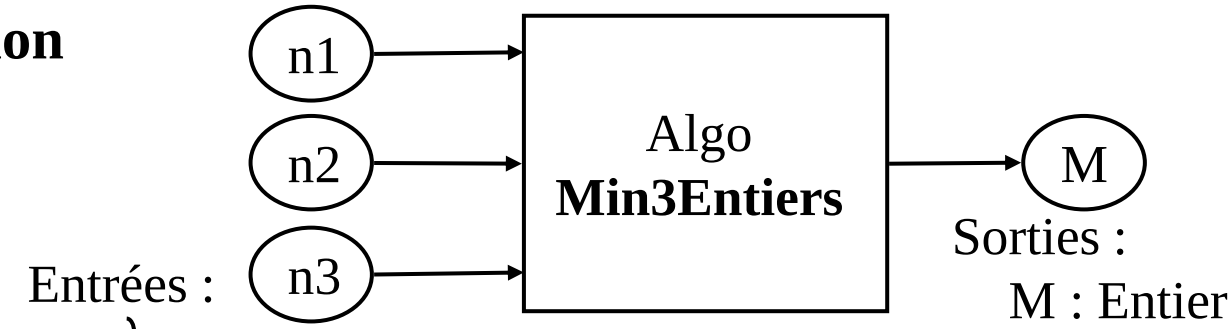
Algorithme

```
Algorithme Disque
    /* déclarations */
    Const  Réel Pi = 3.14
    Réel     R, P, S
/* Corps principal */
Début
    /* Saisie */
    Lire(R) ;
    /* Calculs */
    P ← 2 * PI * R ;
    S ← Pi * R *R ;
    /* Affichage */
    Ecrire("Périmètre = ", P)
    Ecrire("Surface = ", S)
Fin
```

Chap. I : Introduction à l'algorithmique

Algorithme2 : recherche du plus petit parmi trois nombres entiers n1, n2 et n3.

Spécification

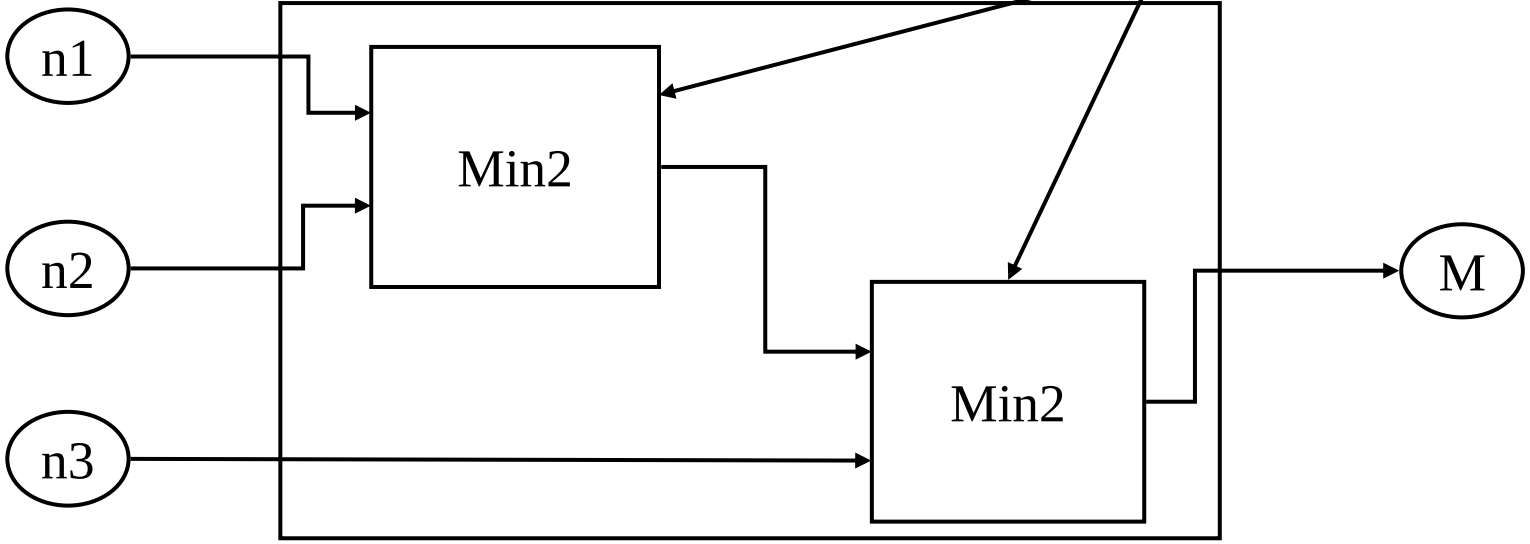


Entrées :
n1, n2, n3 } entiers

Sorties :
M : Entier

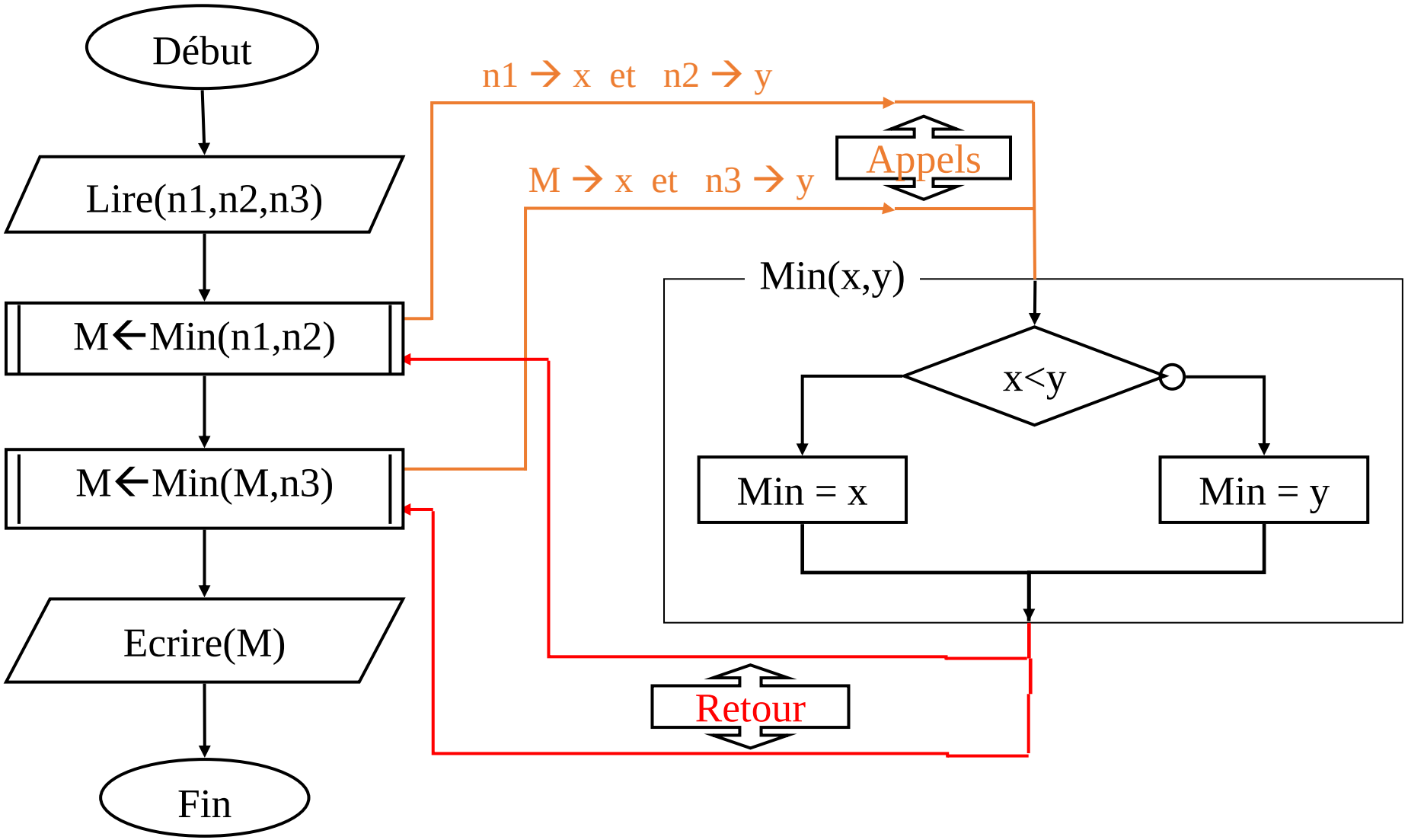
Traitements

Outil pour calculer le minimum de deux nombres. Utilisé dans deux contextes



Remarque : Le problème se trouve simplifié au calcul du minimum de 2 entiers.

Organigramme



Chap. I : Introduction à l'algorithmique

Algorithme

```
Algorithme Min3
/*déclarations */
    Entier n1 , n2 ,n3 , M
/* Définition fonction Min */
Entier Min(Entier x , Entier y)
    Entier m ;
Début
    Si x<y alors
        m ←x
    Sinon
        m ←y
    FinSi
    Retourner m
FinFonction
```

```
/* Corps principal */
Début
    /*introduire n1, n2 et n3 */
    Ecrire("Introduire les nombres : ")
    Lire(n1 , n2 , n3)
    /*  mettre dans M le Min de n2 et n3*/
    M ← Min(n1,n2); /* Appel de Min */
    /* mettre dans M le Min de M et n3*/
    M ← Min(M,n3)
    /* afficher M */
    Ecrire("Le minimum = " , M)
Fin
```

5. Structure d'un algorithme

Un algorithme est structuré en trois parties :

- Un **en-tête** : Pour nommer l'algorithme
- Des **déclarations** : Pour préciser les **données** et les **modules** (**fonctions** et **procédures**)utilisés par l'algorithme
- Un **corps principal** : pour placer la suite des instructions exécutées par l'algorithme

```
Algorithme Nom_Algo
   /* déclarations des objets(données + procédures +fonctions)*/
   …
/* Corps principal */
Début
   /* instructions */
   …
Fin
```