Les bases de l'Informatique

Thierry Vaira

BTS SN

v1.0 - 12 septembre 2016

Qu'est ce qu'un ordinateur ?

- Définition Wikipédia : « Un **ordinateur** est une **machine électronique** qui fonctionne par la lecture séquentielle d'un **ensemble d'instructions**, organisées en **programmes**, qui lui font exécuter des **opérations logiques et arithmétiques** sur des **chiffres binaires**. »
- Le mot « **ordinateur** » fut introduit par le français François Girard chez IBM France en 1955.
- La conception des premières machines fut d'abord mécanique, puis électromécanique et maintenant **électronique**.

Les origines

Historiquement, l'ordinateur prend naissance dans les travaux sur les machines à calculer auxquelles on ajoutera la capacité de programmation :

- La **Pascaline**, une machine à calculer, inventée par le français **Blaise Pascal** en 1642.
- Le **métier à tisser** à base de cartes perforées mis au point par le Lyonnais Joseph Marie **Jacquard** en 1801.
- La **machine à calculer programmable** du britannique **Charles Babbage** en 1834.

Le premier ...

- Le **premier algorithme** écrit par la britannique **Ada Lovelace** en 1843 est considéré comme le premier programme informatique.
- Le **premier qui a mis en œuvre l'algèbre de Boole** (initiée en 1854 par le britannique **George Boole**) est **Claude Shannon**.
- Le **premier programme** jamais exécuté sur un ordinateur à programme stocké en mémoire (une itération, i.e une boucle) est l'œuvre du britannique **David John Wheeler** en 1949.
- Le **premier compilateur** de l'américaine **Grace Murray Hopper** date de 1951. Elle est aussi à l'origine de l'expression de « bug informatique » (un insecte dans un relais).
- Le **premier système d'exploitation** a équipé le **LEO I** en 1951.

Un peu d'histoire

Quelques dates :

- Les premiers ordinateurs datent de 1937 mais ils leurs manquaient souvent des caractéristiques de base pour les considérer comme des « vrais » ordinateurs (l'ASCC/Mark I d'IBM, le Zuse 3, L'ENIAC, le Colossus, l'EDVAC, ...).
- Les bases de l'architecture d'un ordinateur de l'américano-hongrois **John von Neumann** publiées en 1945 sont (encore) utilisées dans la quasi-totalité des ordinateurs.
- Le **microprocesseur** fut inventé en 1969 par l'américain **Ted Hoff** d'Intel pendant le développement d'une calculatrice.
- Le **Micral** inventé par le français **François Gernelle** en 1973 et basé sur un microprocesseur est reconnu comme étant le premier micro-ordinateur de l'histoire.

Qu'est-ce qu'un ordinateur ?

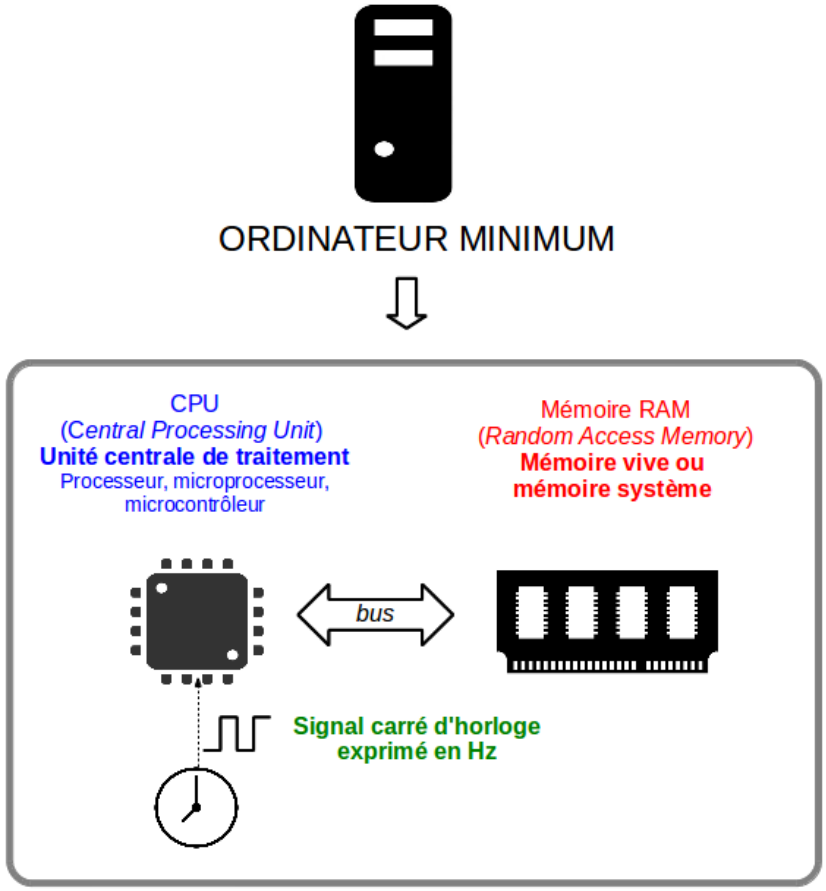
- Les caractéristiques fondamentales d'un ordinateur sont :
  - qu'il soit **électronique** ;
  - qu'il soit **numérique** ;
  - qu'il soit **programmable** ;
  - qu'il puisse **exécuter les quatre opérations élémentaires** (addition, soustraction, multiplication, division) ;
  - qu'il puisse **exécuter des programmes enregistrés en mémoire**.

L'architecture de Von Neumann

- Elle décompose l'ordinateur en quatre parties distinctes :
  - L'**unité arithmétique et logique** (UAL) ou unité de traitement : son rôle est d'effectuer les opérations de base, un peu comme le ferait une calculatrice.
  - L'**unité de contrôle**. C'est l'équivalent des doigts qui actionneraient la calculatrice.
  - La **mémoire** qui contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise entre **mémoire vive** (programmes et données en cours de fonctionnement) et **mémoire permanente** (programmes et données de base de la machine).
  - Les **entrées-sorties** : les dispositifs qui permettent de communiquer avec le monde extérieur.

Un ordinateur minimum

- On peut considérer qu'un « ordinateur minimum » est composé :
  - d'un **processeur** qui exécute les instructions machine des programmes
  - de **mémoire vive** qui contient à la fois les données et le programme
  - éventuellement d'une **horloge** qui cadence et synchronise l'exécution des instructions et l'échange des données



Qu'est-ce qu'un bus ?

- C'est un ensemble de liaisons (fils) entre des composants électroniques qui permet l'échange de bits (0 ou 1) ou de signaux (état haut ou bas). On distingue généralement 3 bus :
  - le **bus de données** de n bits pour échanger les données et le programme. n est un multiple d'une puissance de 2 (8 bits, 16 bits, 32 bits, 64 bits, ...). Ce bus est bidirectionnel.
  - le **bus d'adresse** de n bits pour indiquer directement la position d'une « **case** » (ou **cellule**) dans la mémoire. Le nombre de « cases » mémoires est égale 2^n ($2^{16} = 65536$ adresses soit `64Kio`, $2^{20} = 1048576$ adresses soit `1Mio`, $2^{24} = 16777216$ adresses soit `16Mio`, 2^{32} soit `4Gio` ...). Une « case » mémoire a une **taille d'un octet (8 bits)**. Ce bus est unidirectionnel.
  - le **bus de contrôle ou de commande** pour gérer les deux autres bus et les opérations effectuées (l'horloge pour la synchronisation, un signal pour faire une lecture ou une écriture, ...). Ce bus peut être bidirectionnel.

Qu'est-ce qu'un processeur ?

- En électronique, un processeur est un ensemble de plusieurs milliers, millions ou milliards de **transistors** utilisés comme un interrupteur à deux états et câblés afin de réaliser des opérations (arithmétiques et logiques) et des « cases » mémoires appellées **registres**.

- Un processeur construit en un seul **circuit intégré** est un **microprocesseur** (μP).
- Un **microcontrôleur** (μC) est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires (mémoire morte pour le programme, mémoire vive pour les données), unités périphériques et interfaces d'entrées-sorties.

Et en informatique ?

- En informatique, un processeur est « vu » comme composé :
  - d'une **unité de contrôle** qui se charge de gérer le processeur. Elle peut décoder les instructions, choisir les registres à utiliser, ...
  - des **registres** qui sont des petites mémoires internes très rapides. La taille des registres est de quelques octets et dépend de l'architecture qui correspond généralement au nombre de bits du processeur (un processeur 8 bits aura des registres d'un octet). Les registres généraux servent pour stocker les données dans les échanges avec la mémoire mais aussi pour les résultats des opérations effectuées en interne. Il existe aussi des registres spécialisés (registre d'instruction, registre d'état, ...).
  - d'une ou plusieurs **unités arithmétiques et logiques** (UAL ou ALU, *Aritmetic and Logical Unit*)
  - éventuellement d'une **unité de calcul en virgule flottante** (*Floating-Point Unit*, FPU), qui permet d'accélérer les calculs sur les nombres réels codés en virgule flottante.
  - éventuellement de la **mémoire cache** qui permet d'accélérer les traitements en diminuant les accès à la mémoire vive. Le cache d'instructions reçoit les prochaines instructions à exécuter, le cache de données manipule les données.

Comment « parler » au processeur ?

- Un processeur « parle » le **langage machine** :
  - C'est une suite de bits interprétée par le processeur.
  - C'est le seul langage que le processeur puisse traiter.
  - Il est composé d'**instructions** et de **données** à traiter codées en binaire.
  - Exemple : l'instruction `10110000 01100001` met la valeur hexadécimale 61 dans le registre `AL`. Ici, `10110000` correspond à l'***opcode*** et `01100001` à la **donnée**.
  - Un processeur possède nativement un **jeu d'instructions** composé d'***opcode*** (code opération) et d'opérandes. Il existe des *opcodes* pour faire des opérations arithmétiques, logiques, etc ... Chaque instruction nécessite un certain temps (généralement un nombre de cycles d'horloge) pour s'exécuter.

Comment peut-on « parler » au processeur ?

- Le langage informatique le plus proche du langage machine est l'**assembleur** :
  - ➟ C'est un langage de bas niveau qui représente le langage machine sous une forme lisible par un humain.
  - ➟ Les combinaisons de bits du langage machine sont représentées par des **mnémoniques** (des symboles) faciles à retenir : `ADD` pour l'addition, `MOV` pour la copie de valeurs, etc ...
  - ➟ Exemple : l'instruction binaire `10110000 01100001` sera `movb $0x61,%al` en assembleur.

➟ *Il est en théorie possible de désassembler le code machine d'un programme. En pratique, c'est plus compliqué que cela car il est parfois difficile de savoir si une valeur en mémoire est un opcode ou une donnée. Dans tous les cas, on ne pourra jamais « remonter » plus haut que l'assembleur.*

Comment faire pour « parler » au processeur ?

- Les contraintes sont les suivantes :
  - ➟ L'ordinateur ne sait exécuter qu'un nombre limité d'opérations élémentaires représentées par des instructions codées en binaire
  - ➟ L'ordinateur ne comprend que le langage machine.
- On est donc obliger d'utiliser des « outils » pour traduire un langage simple vers le langage machine.
- On a ensuite créer des langages de programmation évolués (le plus proche possible de l'« humain ») et utilisables sur n'importe quel ordinateur.

➟ *L'arrivée des premiers langages évolués comme le Cobol et le Fortran datent des années 1950.*

Les langages de programmation

- Quel que soit le langage évolué, il faut le **traduire en langage machine**. Il existe des approches différentes :
  - **Assemblage** (assembleur) : traduction du mnémonique (symbole) en *opcode*.
  - **Compilation** (Pascal, C, C++, ...) : traduction de l'ensemble du programme écrit en langage évolué (**code source**) en langage machine (**code objet**). Le programme en langage machine sera ensuite exécuté.
  - **Interprétation** (Javascript, PHP, Python, ...) : traduction de chaque instruction avant de l'exécuter. On nomme souvent ces programmes des **scripts**.
  - **Compilation et Interprétation** (Java) : compilation en un code intermédiaire (*bytecode*) qui n'est pas du code machine mais un code pour une « machine virtuelle ». Ce code est ensuite interprété par un interpréteur (la machine virtuelle).

➠ *Les langages n'ont pas les mêmes performances. On utilise pas les mêmes langages pour faire un OS, un tableur, un site web ou une application pour smartphone.*

Pourquoi faire ?

- Le processeur ne sait que **traiter des données**.
  - ➠ Le traitement est assuré par l'exécution d'instructions simples codées en binaire.
  - ➠ Les données seront n'importe quelle information codée en binaire (une « valeur numérique »).
- On rassemble tout cela à l'intérieur d'un **programme** et finalement :
  - ➠ un programme ne fera que **traiter des données** ...
  - ➠ un ordinateur ne servira qu'à **traiter des données** ...
  - ➠ l'informatique ce n'est que ... **traiter des données** !

Traitons des données I

- Pour manipuler des données, les langages de programmation nous offrent le concept de **variable**.
- Une variable est un **espace de stockage pour un résultat**.
- Une variable est associé à un **symbole** (habituellement un **nom** qui sert d'identifiant) qui renvoie à une **position de la mémoire** (une **adresse**) dont le contenu peut prendre successivement différentes valeurs pendant l'exécution d'un programme.
- De manière générale, les variables ont :
  - un **type** : c'est la convention d'interprétation de la séquence de bits qui constitue la variable. Le type de la variable spécifie aussi sa **taille** (la longueur de cette séquence) soit habituellement 8 bits, 32 bits, 64 bits, ... Cela implique qu'il y a un **domaine de valeurs** (ensemble des valeurs possibles).
  - une **valeur** : c'est la séquence de bits elle même. Cette séquence peut être codée de différrentes façons suivant son type.

Des données numériques

Le caractère '2' :

Caractère '2'

0	0	1	1	0	0	1	0

50 ou 0x32

Code ASCII

L'entier 2 :

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

signe

... 2^3 2^2 2^1 2^0

Le réel 2,0 :

0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

signe **exposant** **mantisse**

La chaîne de caractères "2" :

Caractère '2'

0	0	1	1	0	0	1	0

50 ou 0x32

Code ASCII

Caractère NUL '\0'

0	0	0	0	0	0	0	0

0 ou 0x00

Fin de chaîne

Les types scalaires

- Une **variable scalaire** est destinée par son **type** à contenir une **valeur atomique**.
- Les valeurs atomiques sont :
  - les **booléens**
  - les **nombres entiers**
  - les **nombres à virgule flottante**

Remarque : les **caractères** sont en fait des nombres entiers ! En effet, on est obligé de convertir les caractères sous une forme binaire qui constitue un code. Par exemple, le code ASCII du caractère '`A`' est 65 (`0x41` en hexadécimal).

Les types composés

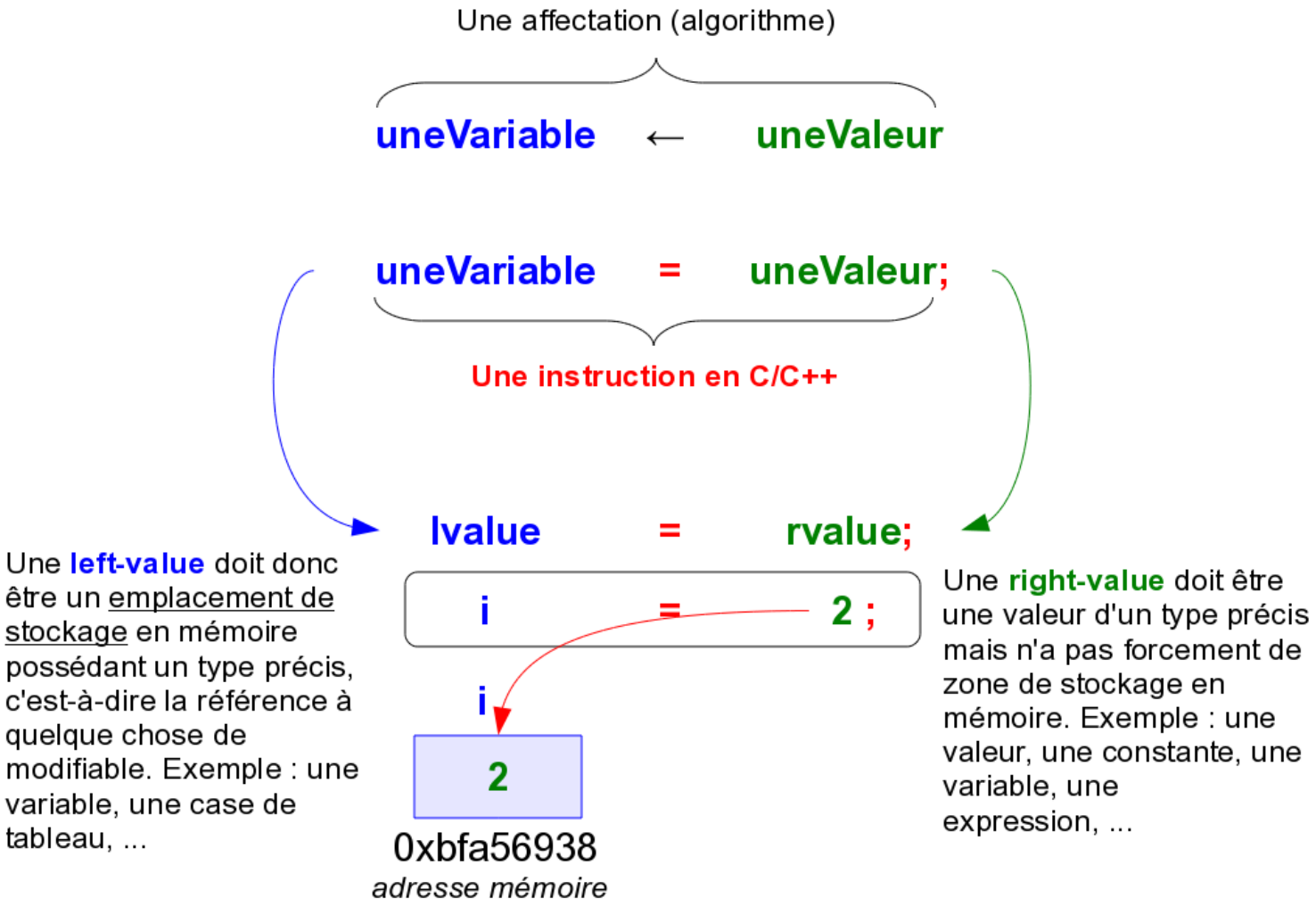
- Il est possible de regrouper des types scalaires dans des **structures de données**.
- Le type composé de base est le **tableau**.
- Il existe aussi parfois la **table associative** (association clé/valeur).
- La **chaîne de caractères** peut être considérée comme un type composé.
- On verra ensuite des **collections** de données (arbre, liste, ...).

Traitons des données II

- Pour traiter des données, il faut pouvoir les manipuler et faire des opérations sur celles-ci.
- Les langages de programmation nous offrent le concept d'**instructions** dans lesquelles on pourra utiliser des **opérateurs**.
- On rappelle que les instructions que l'ordinateur peut comprendre ne sont pas celles du langage humain. Le processeur sait juste exécuter un nombre limité d'instructions bien définies (son jeu d'instructions).
- Des instructions typiques comprises par un ordinateur sont par exemple :
  - copier le contenu de la case 123 et le placer dans la case 456
  - ajouter le contenu de la case 321 à celui de la case 654
  - placer le résultat dans la case 777
  - si le contenu de la case 999 vaut 0, mettre 0 dans la case 345

L'affectation

- La première des instructions est la possibilité d'**affecter une valeur à une variable**. La plupart des langages proposent l'**opérateur d'affectation =** :



L'algorithmie

- L'algorithmie (ou algorithmique), c'est la science qui étudie les algorithmes.
- Un **algorithme** énonce une **résolution d'un problème** sous la forme d'une série d'opérations à effectuer.
- Un algorithme est une suite finie d'instructions élémentaires écrites dans un **langage universel** exécutées de manière séquentielle.
- La mise en œuvre de l'algorithme consiste en l'écriture de ces opérations dans un langage de programmation et constitue alors la brique de base d'un programme informatique.
- Les informaticiens utilisent fréquemment l'anglicisme **implémentation** pour désigner cette mise en œuvre. L'écriture en langage informatique est aussi fréquemment désignée par le terme « **codage** », qui n'a ici aucun rapport avec la cryptographie, mais qui se réfère au terme « **code source** » pour désigner le texte, en langage de programmation, constituant le programme.

L'algorithme

- Un algorithme doit être suffisamment général pour permettre de traiter une classe de problèmes.
- Pour un problème donné, il peut y avoir plusieurs algorithmes ou aucun.
- Pour définir un algorithme, on a besoin d'un langage abstrait, non ambigu et indépendant du langage de programmation
- Exemple du tri d'un jeu de données : il existe de nombreux algorithmes de tri (tri par insertion, tri à bulles, ...) et chacun a ses avantages et inconvénients en fonction du jeu de données.

Méthodologie

➠ Pour résoudre un problème, il faut suivre les quatre étapes suivantes :

- Analyse du problème :
  - comprendre la nature du problème posé
  - préciser les données en "entrées"
  - préciser les résultats que l'on désire obtenir en "sorties"
- Conception d'une solution :
  - déterminer le processus de transformation des données en résultats
  - écrire l'algorithme
- Implémentation de la solution :
  - programmation (« codage ») de l'algorithme dans un langage de programmation.
- Phase de test de la solution :
  - l'objectif d'un test est de détecter une anomalie.
  - sélectionner un jeu de données en "entrées"
  - définir le résultat attendu en "sorties"
  - vérifier le résultat obtenu avec le résultat attendu

Les variables dans les algorithmes

- Pour créer une variable, il faut la **déclarer** en lui donnant **un nom** (éloquent et précis) et **un type** :

```
// Déclaration d'une variable de type entier
Variable indice : Entier

// Déclaration d'une variable de type caractère
Variable lettre : Caractère

// Déclaration d'une constante de type réel
Constante PI : Réel = 3,14

// On affacte une valeur à une variable
indice <- 0
```

Les variables dans les langages de programmation

- Dans les langages compilés, les variables doivent être déclarées en précisant leur type. Il est conseillé de toujours les initialiser :

```
// En C/C++ :

// une variable entière
int i = 0;
// une variable réelle
float j = 2.5;
```

- Dans les langages interprétés, seules les valeurs ont un type ce qui n'oblige pas toujours à déclarer les variables :

```
// En PHP :

// une valeur entière
$i = 0;
// une valeur réelle
$i = 2.5;
```

Algorithme vs Programme

Un algorithme

```
Variable x,y,z : Entier

Début
   Ecrire "Saisir deux valeurs
        entières : "
   Lire x
   Lire y
   z <- x + y
   Ecrire "Résultat : "
   Ecrire z
Fin
```

Son implémentation en C++

```
int main()
{
   int x, y, z;

   cout << "Saisir deux valeurs
        entières : ";
   cin >> x;
   cin >> y;
   z = x + y;
   cout << "Résultat : ";
   cout << z;

   return 0;
}
```

Les structures fondamentales

➠ Un programme comporte deux types d'instructions :

- Les **instructions de base**
  - Elle permettent de manipuler les variables : affectation, lecture, écriture
  - Elle permettent de faire des opérations : addition, ...
- Les **instructions de structuration**
  - Elles servent à préciser comment doivent s'enchaîner chronologiquement les instructions de bases
  - Elles sont de deux types : structure conditionnelles et structures itératives (les boucles).

Les structures conditionnelles

```
Si <condition> Alors
   instruction(s)
[Sinon instruction(s)]
FinSi
```

```
Si (a>0) Alors
   Ecrire "a est positive"
Sinon
   Ecrire "a est négative"
FinSi
```

- La condition est une expression logique (un booléen : VRAI/FAUX)
- On peut combiner plusieurs tests avec des ET (&&), OU (||) ou utiliser la NEGATION (!)
- La partie « Sinon » est facultative
- On peut imbriquer plusieurs structures conditionnelles

Les choix multiples

➠ Lorsque que l'on souhaite conditionner l'exécution de plusieurs ensembles d'instructions par la valeur que prend une variable, plutôt que d'utiliser des structures conditionnelles imbriquées, on peut utiliser un `Selon` (un `switch` en C/C++) :

```
Selon <identificateur>
   valeur_1 : instructions
   valeur_2 : instructions
   ...
   valeur_n : instructions
   [autres : instructions]
FinSelon
```

```
Selon op
   "s" : Ecrire "Opération somme"
   "p" : Ecrire "Opération produit"
   autres : Ecrire "Erreur : l'
        opération est inconnue !"
FinSelon
```

Les structures répétitives

➠ Les structures répétitives permettent d'**itérer une instruction ou une suite d'instructions**. En programmation, on parle de **« boucle »**.

- Les instructions sont répétées `0` **à *n* fois** (*n* est un nombre de fois indéterminé qui dépend uniquement de la condition de sortie de la boucle)

  `TantQue` **condition** `Faire` **instruction(s)** `FinTantQue`

- Les instructions sont répétées `1` **à *n* fois** (*n* est un nombre de fois indéterminé qui dépend uniquement de la condition de sortie de la boucle)

  `Répéter` **instruction(s)** `TantQue` **condition**

- Les instructions sont répétées ***n* fois** (*n* est un nombre de fois déterminé)

  `Pour` **variable** `de ... à ... Faire` **instruction(s)** `FinPour`

➟ Le premier programme jamais exécuté sur un ordinateur à programme stocké en mémoire (l'EDSAC) est un exemple d'itération. Il a été écrit et exécuté par David Wheeler au laboratoire informatique de Cambridge le 6 mai 1949 pour calculer et afficher une simple liste de carrés. Le code devait ressembler à ceci :

La boucle TantQue

```
Variable n,a,c : Entier
Début
   n <- 0
   TantQue (n <> 50) Faire
      Lire a
      c <- a x a
      Ecrire c
      n <- n + 1
   FinTantQue
   Ecrire "Fin du programme"
Fin
```

Son implémentation en C++

```
int main()
{
   int n, a, c;
   n = 0;
   while (n != 50)
   {
      cin >> a;
      c = a * a;
      cout << c << endl;
      n = n + 1;
   }
   cout << "Fin du programme";
   return 0;
}
```

➟ Le même programme :

La boucle Pour

```
Variable n,a,c : Entier

Début
   Pour n de 0 à 49 Faire
      Lire a
      c <- a x a
      Ecrire c
   FinPour
   Ecrire "Fin du programme"
Fin
```

Son implémentation en C++

```
int main()
{
   int n, a, c;

   for(n=0;n!=50;n++)
   {
      cin >> a;
      c = a * a;
      cout << c << endl;
   }
   cout << "Fin du programme";
   return 0;
}
```

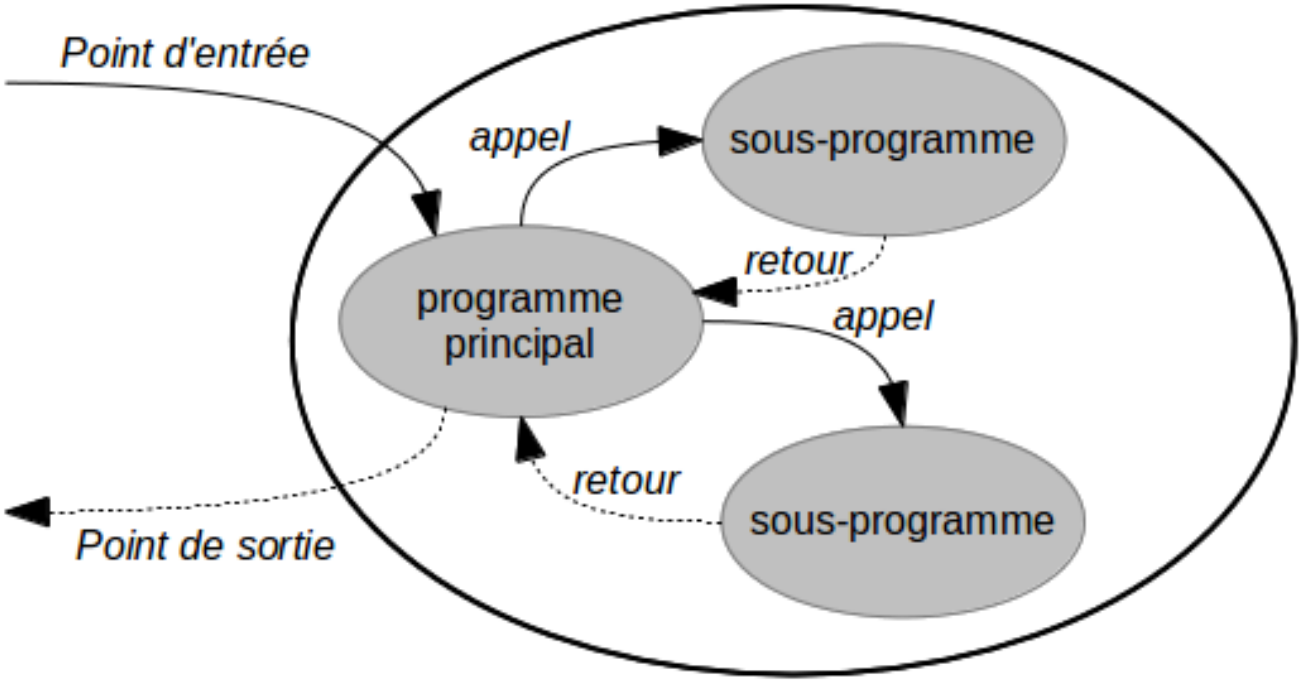
L'approche fonctionnelle

- Décomposer un problème en sous problèmes :
  - Ceci conduit souvent à diminuer la complexité d'un problème et permet de le résoudre plus facilement.
- Éviter de répéter plusieurs fois les mêmes lignes de code :
  - Ceci facilite la résolution de bogues mais aussi le processus de maintenance.
- Généraliser certaines parties de programmes :
  - La décomposition en module permet de constituer des sous-programmes réutilisables dans d'autres contextes.

Structure d'un programme

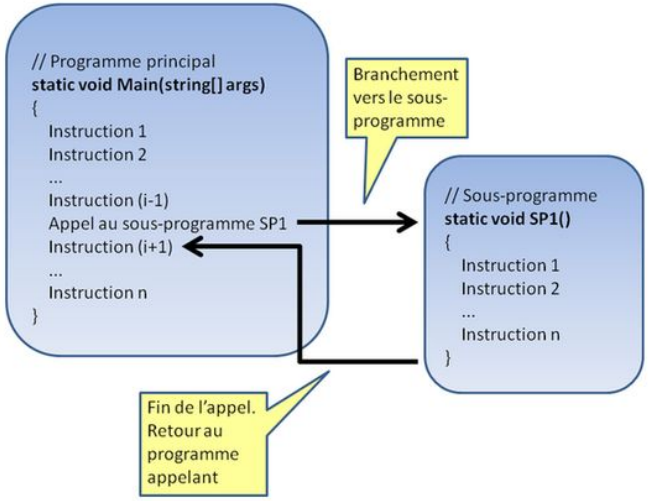
➠ Dans le cas d'une approche fonctionnelle, un programme n'est plus une simple séquence d'instructions mais est constitué :

- D'un ensemble de **sous-programmes** et
- D'un et un **seul programme principal** : unique et obligatoire.



Déroulement d'un programme

- L'exécution du programme commence par l'exécution du programme principal
- L'appel à un sous programme permet de déclencher son exécution, en interrompant le déroulement séquentiel des instructions du programme principal
- Le déroulement des instructions du programme reprend, dès que le sous programme est terminé, à l'instruction qui suit l'appel



Les sous-programmes

➟ On distingue deux types de sous-programmes :

- Les **fonctions**
  - Sous-programme qui retourne **une et une seule valeur** : permet de ne récupérer qu'un résultat.
  - Par convention, ce type de sous-programme ne devrait pas interagir avec l'environnement (écran, utilisateur).
- Les **procédures**
  - Sous-programme qui permet de récupérer de **0 à *n* résultats**
  - Par convention, ce type de sous-programme peut interagir avec l'environnement (écran, utilisateur).
- Cette distinction ne se retrouve pas dans tous les langages de programmation !
  - Par exemple, le C/C++ n'admet que le concept des fonctions qui serviront à la fois pour les fonctions et les procédures.

Définir une fonction

Algorithme :

```
Fonction poserQuestion(q : Chaîne de
     caractères) : Caractère
Donnée(s) : q la question
Résultat : Lit la réponse et retourne
     'V' pour vrai, sinon 'F' pour faux
Variable locale r : Caractère

Début
    Ecrire q
    Répéter
        Ecrire "(V)rai ou (F)aux ?"
        Lire r
    TantQue (r<>'V' ET r<>'F')
    Retourner r
Fin
```

En C++ :

```
char poserQuestion(string q)
{
    char r;

    cout << q;
    do
    {
        cout << "(V)rai ou (F)aux ? ";
        cin >> r;
    }
    while (r!='V' && r!='F');
    return r;
}
```

Appeler une fonction

Algorithme :

```
...
score <- 0
question <- "1. ADN signifie Anti-
     Démangeaison-Nasale ?"
reponse <- poserQuestion(question)
Si reponse = 'F'
    Alors score <- score - 1
    Sinon score <- score + 1
FinSi

question <- "2. La programmation c'est
     facile ?"
...
```

En C++ :

```
...
score = 0;
question = "1. ADN signifie Anti-
     Démangeaison-Nasale ?";
reponse = poserQuestion(question);
if (reponse == 'F')
{
   score = score - 1;
}
else
{
   score = score + 1;
}

question = "2. La programmation c'est
     facile ?";
...
```

Bibliothèque logicielle

- Une **bibliothèque logicielle** est un **ensemble de fonctions** utilitaires, regroupées et mises à disposition afin de pouvoir être utilisées sans avoir à les réécrire.
- Les bibliothèques logicielles ne sont pas complètement des « exécutables » car elles ne possèdent pas de programme principal et par conséquent ne peuvent pas être exécutées directement.
- Les bibliothèques logicielles sont :
  - une interface de programmation (API, *Application Programming Interface*) ;
  - les composants d'un kit de développement logiciel (SDK, *Software Developement Kit*) ;
  - parfois regroupées en un *framework*, de façon à constituer un ensemble cohérent et complémentaire de bibliothèques.
- Exemples :
  - Fichier `.DLL` (*Dynamic Link Library*) ou Bibliothèque de liens dynamiques (Windows).
  - Fichier `.so` (*Shared Object*) ou Bibliothèque dynamique (Linux).

L'approche orientée objet

- Décomposer un problème en objets et les faire interagir entre eux :
  - Un **objet** est caractérisé par le rassemblement, au sein d'une même **unité d'exécution**, d'un ensemble de **propriétés** (constituant son **état**) et d'un **comportement**
  - La notion de **propriété** est matérialisée par un **attribut** qui est une variable locale à l'objet
  - La notion de **comportement** est matérialisée par un ensemble de **méthodes** qui sont ses sous-programmes
  - La **classe** est le modèle (le « moule ») pour créer des objets logiciels.
- On distinguera les langages capables de faire la programmation orientée objet (POO) :
  - C : approche fonctionnelle seulement
  - C++, Java, PHP5, ... : langages orienté objet

Exemple d'objet : une lampe

- Une lampe est **caractérisée par** :
  - Sa **puissance** (une **propriété** ➠ un **attribut**)
  - Le **fait qu'elle soit allumée ou éteinte** (un **état**)
- Au niveau **comportement**, les **actions possibles** sur une lampe sont :
  - L'**allumer** (une **méthode**)
  - L'**éteindre** (une autre **méthode**)

Coder une classe

En Java :

```java
public class Lampe
{
    private int puissance;
    private boolean estAllumee;

    public void allumer()
    {
        this.estAllumee = true;
    }
    public void eteindre()
    {
        this.estAllumee = false;
    }
}
```

En C++ :

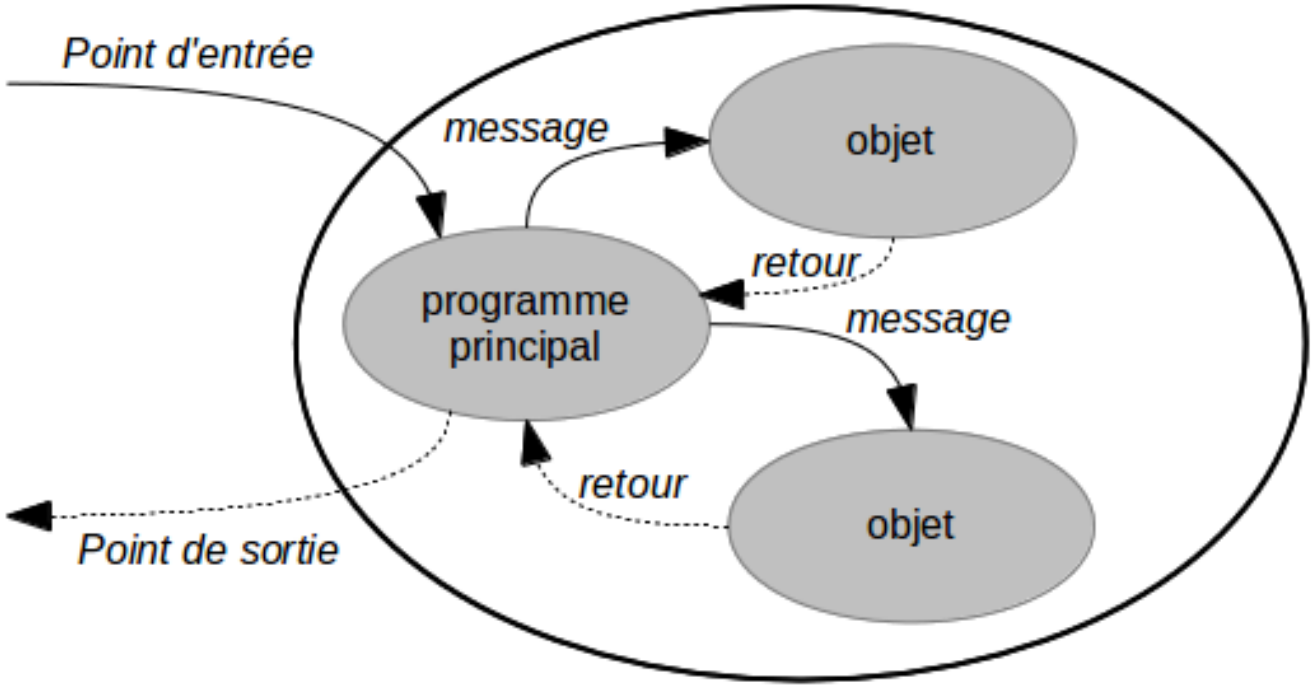
```cpp
class Lampe
{
    private:
        int puissance;
        bool estAllumee;

    public:
        void allumer()
        {
            this->estAllumee = true;
        }
        void eteindre()
        {
            this->estAllumee = false;
        }
};
```

La Salle Avignon

Structure d'un programme orienté objet

➠ Dans le cas d'une approche orientée objet, un programme n'est plus une simple séquence d'instructions mais est constitué :

- D'un ensemble d'**objets** s'échangeant des **messages** (i.e. appel d'une méthode) et
- D'un et un **seul programme principal** : unique et obligatoire.



Rob Pike (ancien chercheur des Laboratoires Bell et maintenant ingénieur chez Google) :

> *« Règle n°4 : Les algorithmes élégants comportent plus d'erreurs que ceux qui sont plus simples, et ils sont plus difficiles à appliquer. Utilisez des algorithmes simples ainsi que des structures de données simples. »*

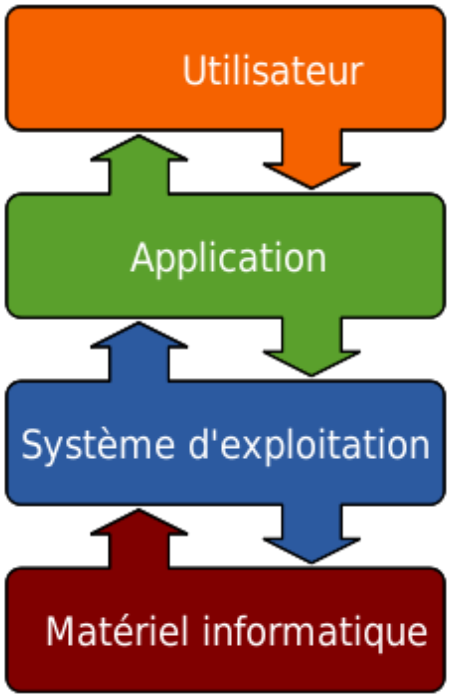
➠ Cette règle n°4 est une des instances de la philosophie de conception KISS (*Keep it Simple, Stupid* dans le sens de « Ne complique pas les choses »).

> *« Règle n°5 : Les données prévalent sur le code. Si vous avez conçu la structure des données appropriée et bien organisé le tout, les algorithmes viendront d'eux-mêmes. La structure des données est le coeur de la programmation, et non pas les algorithmes. »*

➠ Cette règle n°5 est souvent résumée par « Écrivez du code stupide qui utilise des données futées ! ».

Qu'est-ce qu'un OS ?

- Un **système d'exploitation** (SE ou **OS** pour *operating system*) est un ensemble de programmes d'un équipement informatique qui sert d'**interface** entre le matériel et les logiciels applicatifs.
- C'est donc une **couche logicielle** (*software*) qui permet et coordonne l'utilisation du matériel (*hardware*) entre les différents programmes d'application.
- Un système d'exploitation est typiquement composé : d'un **noyau** (*kernel*), de bibliothèques, d'un ensemble d'outils système et de programmes applicatifs de base.



Est-ce indispensable ?

- **Non !** Il y a de nombreux ordinateurs qui ne possèdent pas de système d'exploitation.
- Le programme s'exécute alors directement sur la machine par contre il a la charge de gérer le matériel.
- C'est souvent le cas dans les **systèmes embarqués** :

  Un système embarqué est défini comme un **système électronique et informatique autonome spécialisé dans une tâche bien précise**. Le terme désigne aussi bien le matériel informatique que le logiciel utilisé. Ses ressources sont généralement limitées. Cette limitation est généralement d'ordre spatial (encombrement réduit) et énergétique (consommation restreinte).

Architecture

- Avec un système d'exploitation, les programmes ne s'exécutent pas directement mais sont pris en charge par l'OS.
- C'est donc que le système d'exploitation qui :
  - coordonne l'utilisation du ou des processeur(s), et accorde un certain temps pour l'exécution de chaque **processus** (un **programme en cours d'exécution**)
  - réserve de l'espace dans les mémoires pour les besoins des processus
  - permet l'accès aux **fichiers et aux répertoires**
  - reçoit les manipulations effectuées par l'utilisateur via le clavier, la souris ou d'autres périphériques, et les transmet aux différents processus
- Certains de ces services sont fournis par une **interface de programmation** (*System Calls* pour Unix et *WIN32* pour Windows).
- Une **couche d'abstraction matérielle HAL** (*Hardware Abstraction Layer*) est chargée de masquer les particularités matérielles puis l'OS intègre des **pilotes de périphériques** (*drivers*) pour exploiter le matériel spécifique installé sur la machine.

Quelques remarques

- De nombreux logiciels applicatifs sur le marché sont construits pour fonctionner avec un système d'exploitation en particulier, ou une famille en particulier.
- Un système d'exploitation est construit pour fonctionner avec une gamme de machines donnée (type de processeur, constructeur, architecture).
- Pour l'acheteur le choix de la famille de machine limite le choix du système d'exploitation, qui lui-même limite le choix des logiciels applicatifs.
- L'utilité d'un système d'exploitation pour l'usager accroît avec le nombre de logiciels applicatifs qui sont prévus pour lui.
- La popularité élevée d'un système d'exploitation attire les éditeurs de logiciels applicatifs, ce qui accroît encore sa popularité. Ce phénomène fait que le marché est sujet aux situations de monopole.
- Les systèmes d'exploitation sont souvent vendus avec les appareils informatiques.

Quel est le marché ?

- En 2010, les deux familles de systèmes d'exploitation les plus populaires sont **Unix** (dont Mac OS X et Linux) et **©Windows**.
- La gamme des systèmes **Windows** équipe aujourd'hui 38 % des serveurs et 90 % des ordinateurs personnels, ce qui la place en situation de monopole notamment auprès du grand public. En 2008, ses parts de marché sont descendues en dessous de 90 % pour la première fois depuis 15 ans.
- La famille de systèmes d'exploitation **Unix** compte plus de 25 membres et ses parts de marché sont de presque 50 % sur les serveurs. La famille **Unix** anime 60 % des sites web dans le monde et **Linux** équipe 95 % des 500 super-ordinateurs du monde et la majorité des *box* et routeurs Internet.
- Le système d'exploitation le plus répandu dans les *smartphones* et tablettes est un **Linux (Android)**.
- On dénombre plus d'une centaine de systèmes d'exploitation dans le monde.

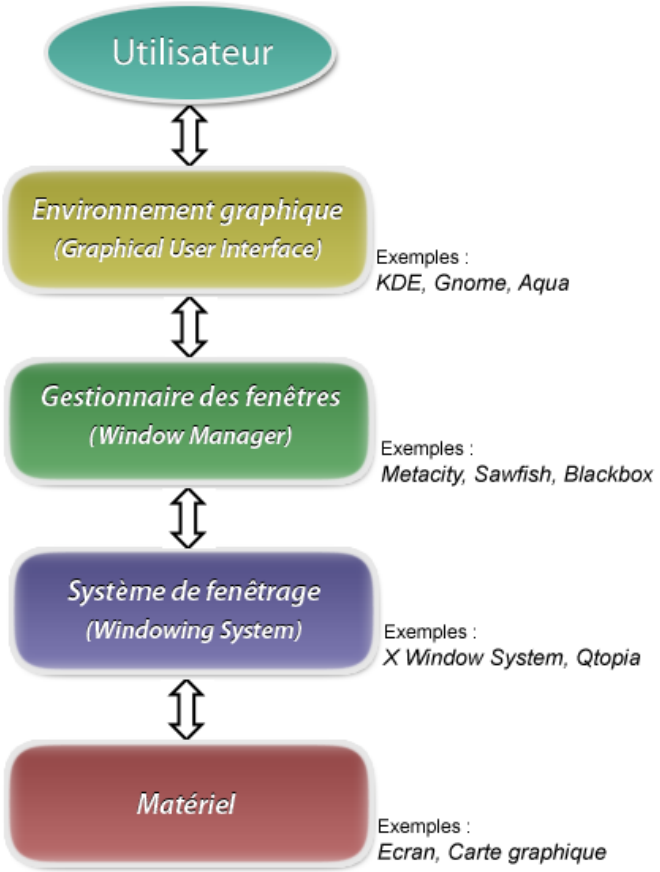
- Un système d'exploitation est dit **multi-tâches** quand il permet l'exécution simultanée de plusieurs programmes. Tous les systèmes d'exploitation actuels sont multi-tâches.
- Il est dit **multi-utilisateurs** quand il est conçu pour être utilisé simultanément par plusieurs usagers, souvent à travers un réseau informatique (notion de serveurs). Ils sont multi-tâches et en général sécurisés, c'est-à-dire qu'il vont refuser d'exécuter toute opération pour laquelle l'usager n'a pas préalablement reçu une permission.
- Il est dit **multi-processeurs** quand il est conçu pour exploiter un ordinateur équipé de plusieurs processeurs. Dans de tels systèmes d'exploitation, plusieurs programmes sont exécutés simultanément par les différents processeurs.
- Il est dit **temps réel** quand il garantit que les opérations seront effectuées en respectant des délais stricts, et ce quelles que soient les conditions d'utilisation (charge du système). De tels systèmes d'exploitation sont utilisés dans l'industrie, l'aéronautique ou l'électronique pour créer des systèmes temps réel (souvent embarqué).

IHM

- L'**IHM** (**Interface Homme-Machine**) permet à un utilisateur de dialoguer avec la machine. On distingue deux types d'IHM :
  - **GUI** (*Graphical User Interface*) ou « **interface utilisateur graphique** » : les parties les plus typiques de ce type d'environnement sont le pointeur de souris, les fenêtres, le bureau, les icônes, les boutons, les menus, les barres de défilement, ... Les systèmes d'exploitation grand public (Windows, MacOS, GNU/Linux, etc.) sont pourvus d'une interface graphique qui, dans un soucis d'ergonomie, se veut conviviale, simple d'utilisation et accessible au plus grand nombre pour l'usage d'un ordinateur personnel.
  - **CLI** (*Command Line Interface*) ou « **interface en ligne de commande** » est encore utilisée en raison de sa puissance, de sa grande rapidité, son uniformité, sa stabilité et du peu de ressources nécessaires à son fonctionnement. Le système d'exploitation permet cette possibilité par l'intermédiaire d'un interpréteur de commandes (le *shell*). Beaucoup de serveurs ne s'administrent qu'en ligne de commande.
- Le ***shell*** (coquille) est la partie la plus externe du système d'exploitation, c'est l'interface utilisateur du système d'exploitation. Ce terme est surtout utilisé dans la famille UNIX.

Environnement fenêtré

- Aussi appelé **WIMP** (*Windows* (fenêtres), *Icons* (icônes), *Menus* (menus) and *Pointing device* (dispositif de pointage)), ce type d'interface graphique a été inventé par la firme **Xerox**, puis copié et rendu célèbre par le **Macintosh** ensuite copié et popularisé par **Windows**.
- Les parties les plus typiques d'un environnement fenêtré sont le concept de bureau.
- Sous GNU/Linux aujourd'hui, les environnements de bureau regroupent un environnement graphique et un gestionnaire de fenêtres : GNOME, KDE, Xfce, GNUstep, CDE, ... Mac OS X utilise Aqua et sous Windows c'est Aero.



Le *shell*

- En général, le *shell* permet :
  - l'exécution de commandes
  - la redirection des entrées et des sorties
  - la gestion des variables d'environnement
  - la possibilité de réaliser des **scripts** pour l'automatisation de tâches
- Il existe de nombreux *shell* sous UNIX. Sous GNU/Linux, le *shell* par défaut est `bash` (*Bourne Again Shell*).
- `cmd.exe` est l'interpréteur de commande en mode texte de Windows. Il peut interpréter des fichiers *batch* (`.BAT` ou `.CMD`) qui sont des scripts contenant une série de commandes. Maintenant, Windows fournit aussi le *PowerShell*. Sous Windows, l'appellation « *shell* » regroupe aussi l'interface graphique, en général l'*Explorer*.

Processus

- Un programme qui s'exécute est appelé un **processus**.
- Un processus comporte du code machine exécutable, une zone mémoire (données allouées par le processus), une pile ou *stack* (pour les variables locales des fonctions et la gestion des appels et retour des fonctions) et un tas ou *heap* pour les allocations dynamiques (fonctions `malloc` ou `new`).
- Un processus est créé avec 3 flux standards : un flux d'entrée (***stdin***) relié par défaut au clavier, un flux de sortie (***stdout***) relié par défaut à l'écran et un flux d'erreur (***stderr***).
- Un **flux ou flot** est un **canal recevant ou fournissant de l'information**.
- Ce processus est une entité qui, de sa création à sa mort, est identifié par l'OS par une valeur numérique : le **PID** (*Process IDentifier*).
- Les commandes `ps` et `top` listent les processus sous UNIX et, sous Windows on utilisera le gestionnaire de tâches (`taskmgr.exe`).

Qu'est-ce qu'un système de fichiers ?

- Un **système de fichiers** (*file system*) définit l'**organisation d'une partition d'un support de stockage**.
- C'est une **structure de données** permettant de stocker les informations et de les organiser dans des fichiers sur ce que l'on appelle des mémoires secondaires (disque dur, CD-ROM, etc.).
- Un système de fichiers offre à l'utilisateur une vue abstraite sur ses données (**fichiers**) et permet de les localiser à partir d'un **chemin d'accès** (*path*) dans une **arborescence de répertoires** (dossier).
- Le fichier est la plus petite entité logique de stockage sur un disque.
- Une partition est une partie d'un disque dur destinée à accueillir un système de fichiers.
- Le **formatage** prépare une partition d'un support de données de stockage en y inscrivant un système de fichiers, de façon à ce qu'il soit reconnu par le système d'exploitation de l'ordinateur.
- Il existe de nombreux systèmes de fichiers différents : FAT, NTFS, HFS, ext2, ext3, UFS, etc.

Notions de fichier

- Un **fichier** est une **suite d'octets** portant un nom et conservé dans une mémoire.
- Le contenu du fichier peut représenter n'importe quelle donnée binaire : un programme, une image, un texte, etc.
- Les fichiers sont classés dans des groupes appelés **répertoires**, chaque répertoire peut contenir d'autres répertoires, formant ainsi une **organisation arborescente**.
- Les fichiers sont la plupart du temps conservés (stockés) sur des **mémoires de masse** tels que les disques durs. Des systèmes de fichiers existent aussi pour la mémoire RAM.
- Dans un système d'exploitation multi-utilisateurs, les programmes qui manipulent le système de fichiers effectuent des **contrôles d'accès** (notion de **droits**).
- Les données descriptives (métadonnées) comme les dates de création et de modification, le propriétaire du fichier ainsi que les droits d'accès sont conservés dans le système de fichiers lui-même.

Notions de système de fichiers

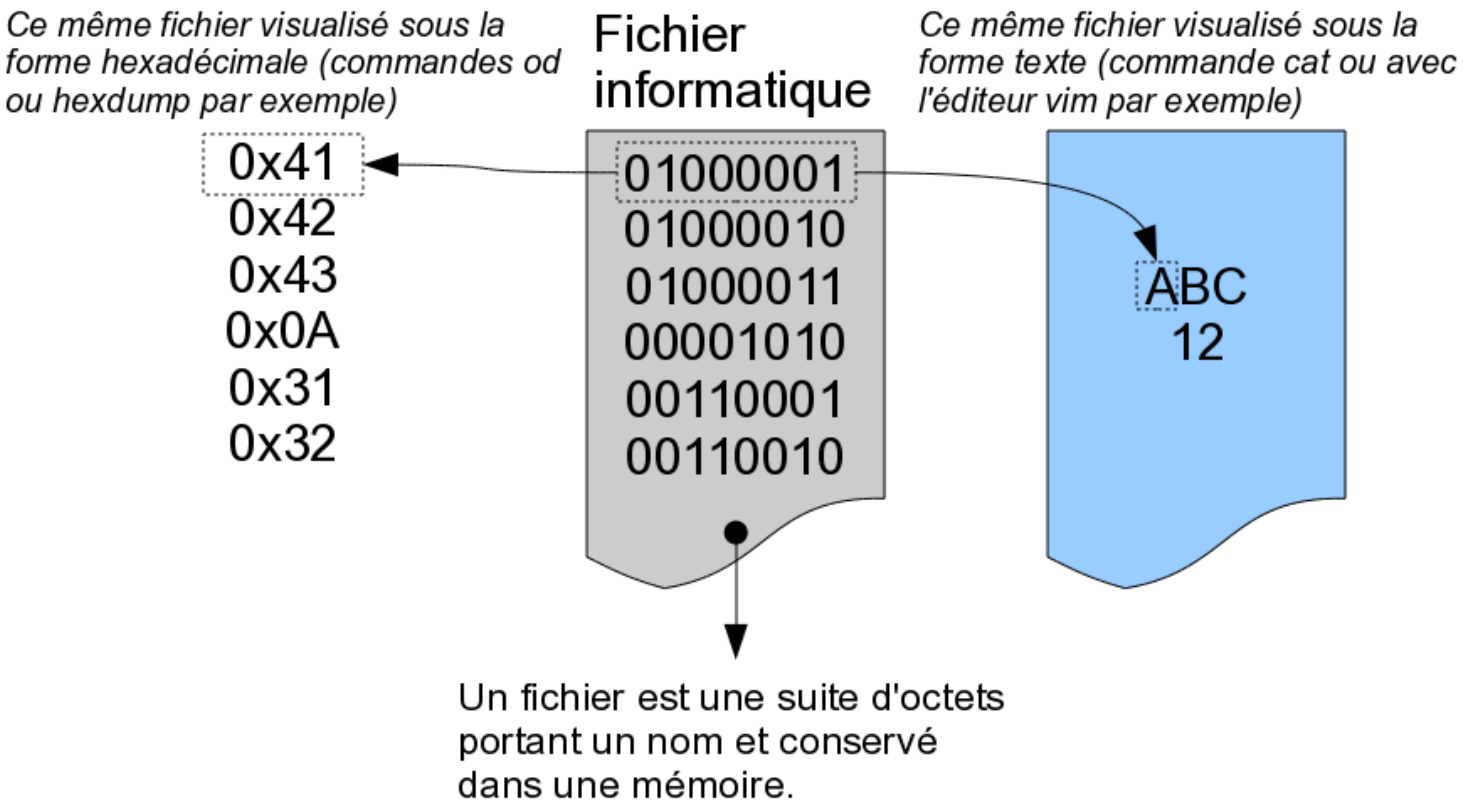
- Un système de fichiers découpe l'espace d'une partition en **blocs d'allocation** de taille fixe (paramétrable). Ceci induit que la taille physique (réellement occupée) est légèrement supérieure à la taille du fichier (exprimée en octets).
- L'**extension** d'un fichier, suffixe ajouté au nom du fichier pour indiquer la nature de son contenu, est seulement destinée à l'utilisateur. L'usage des extensions est une pratique généralisée sur **Windows** et une pratique courante sur **Unix**.
- Chaque fichier est vu par le système de fichiers de plusieurs façons :
  - un **descripteur** de fichier (souvent un entier unique) permettant de l'identifier au niveau système
  - une entrée dans un répertoire permettant de le situer et de le nommer
  - des métadonnées sur le fichier permettant de le définir et de le décrire
  - un ou plusieurs blocs (selon sa taille) permettant d'accéder aux données du fichier (son contenu)

Types de fichiers

- Les fichiers « **texte** » ont un contenu pouvant être interprété comme du texte (une suite de bits représentant un caractère), la plupart du temps en **codage ASCII**.
- Les fichiers « **binaire** » respectent un **format de fichier** (convention normalisée ou non) utilisé pour représenter et stocker des données.
- Un fichier (texte ou binaire) qui a subi une transformation par un algorithme en vue de diminuer sa taille est appelé **fichier compressé**. Le fichier transformé est un fichier binaire.
- Une **archive** est un fichier (binaire) dans lequel se trouve regroupé des fichiers ou tout le contenu d'une arborescence. Le but principal d'une archive est de tout contenir (données + descriptions) en un seul fichier. Les archives sont souvent compressés.

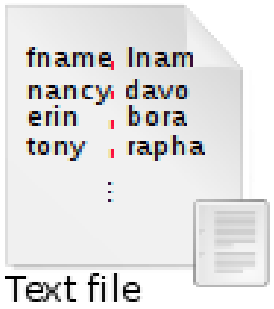
Les fichiers « texte »

- Les fichiers « **texte** » ont un contenu pouvant être interprété comme du texte (une suite de bits représentant un caractère), la plupart du temps en **codage ASCII**. On utilise souvent un **éditeur de texte** pour les manipuler. Exemples : code source, fichiers de configuration, ...



Les séparateurs

- Si les fichiers « **texte** » ne sont qu'une suite de caractères encodés, il est tout de même possible de les structurer.
- Pour cela, on utilise des **séparateurs** (*flag*) ou, délimiteur ou marqueur.
- Un séparateur est une **séquence de un ou plusieurs caractères** servant à délimiter la frontière entre différentes zones (**champs**) de texte ou autres flux de données.



- On utilise généralement :
  - de simples caractères ASCII comme la virgule ',', les deux points ':', le point-virgule ';', etc ...
  - des caractères de contrôle du code ASCII : la tabulation '\t', le changement de ligne '\n', le retour chariot '\r', etc ...

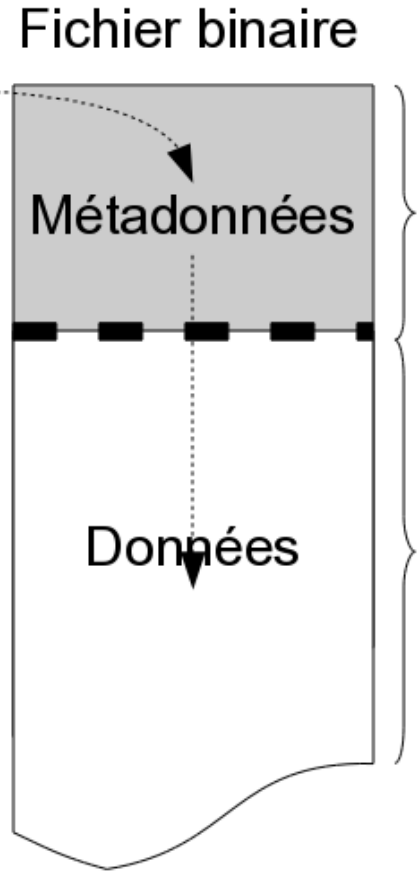
Les fichiers « binaire »

- Les fichiers « **binaire** » correspondent souvent à un **format** précis lié à un logiciel applicatif spécifique. Exemples : code machine (exécutable), fichiers multimédias (images, sons, vidéos, traitement de texte, etc.). Tout ce qui n'est pas un fichier texte est un fichier binaire !

Il faut connaître la structure des métadonnées pour les extraire et les analyser et ensuite pouvoir accéder et exploiter les données proprement dites du fichier.

Les fichiers binaires ont donc un « format », c'est-à-dire une structure propre.

Le programme qui manipule un type de fichier binaire « connaît » ce format.

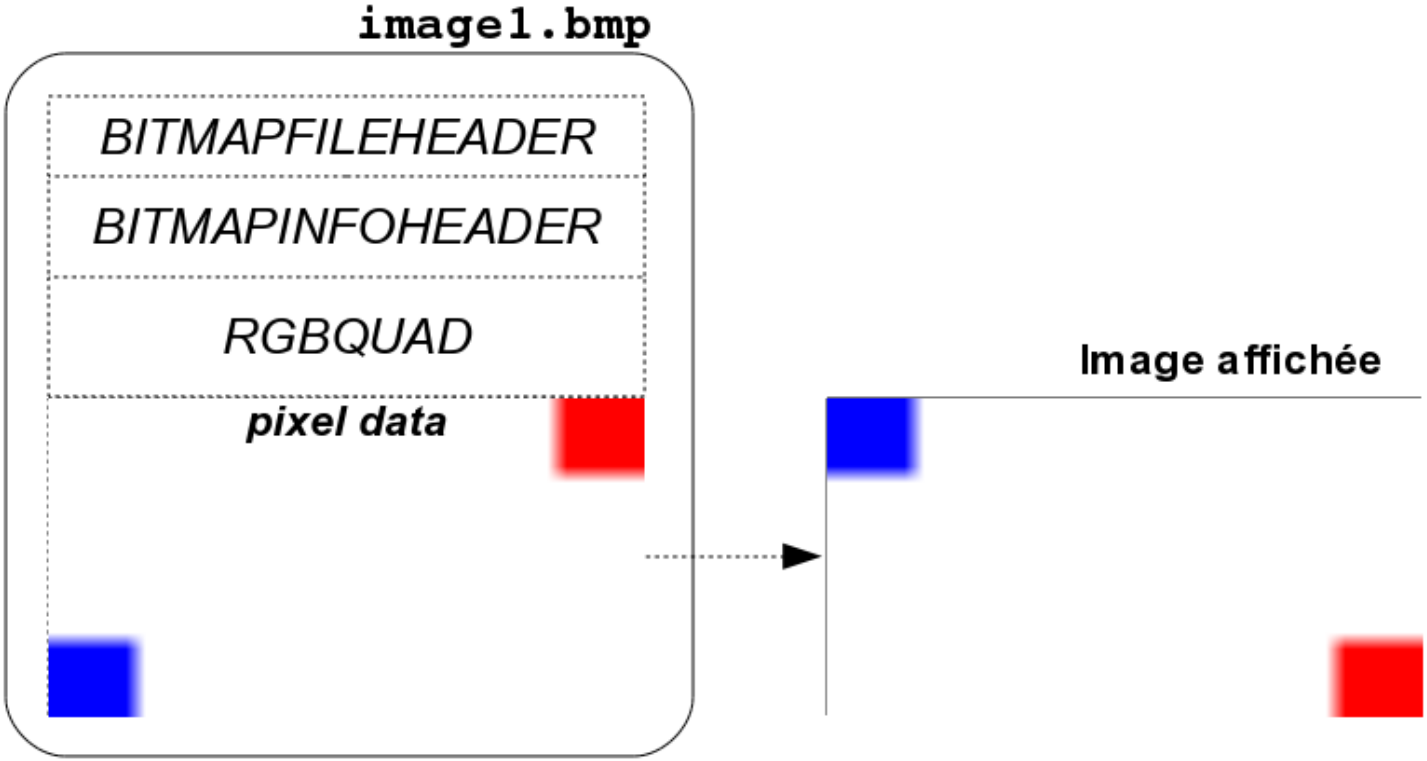


Une métadonnée est une donnée servant à définir ou décrire une autre donnée.

Les données sont codées sous forme de séquence de bits pour représenter des informations de texte, d'image, de son, etc.

Exemple de fichier binaire

- L'images BMP sont stockées dans des fichiers « **binaire** » qui respectent un format de fichier utilisé pour représenter et stocker les données (les *pixels*, des points en couleurs).



Fichier BMP

Champ	Valeur en hexadécimale	Valeur décodée
Type (`bfType`) en ASCII	**42 4d (code ascii)**	**BM**
Taille (`bfSize`) en octets	B6 07 00 00 donc 00 00 07 B6	**1974**
Offset (`bfOffBits`) en octets	36 00 00 00 donc 00 00 00 36	**54**

Champ	Valeur en hexadécimale	Valeur décodée
Taille de cette entête (`biSize`) en octets	28 00 00 00 donc 00 00 00 28	**40**
Largeur (`biWidth`) en pixels	20 00 00 00 donc 00 00 00 20	**32**
Height (`biHeight`) en pixels	14 00 00 00 donc 00 00 00 14	**20**
Codage des couleurs (`biBitCount`) en bits	18 00 donc 00 18	**24**
Taille des données image (`biSizeImage`) en octets	80 07 00 00 donc 00 00 07 80	**1920**
Nb pixels par mètre en X (`biXPelsPerMeter`)	13 0b 00 00 donc 00 00 0b 13	**2835**
Nb pixels par mètre en Y (`biYPelsPerMeter`)	13 0b 00 00 donc 00 00 0b 13	**2835**