

Les bases de l'Informatique L'algorithmie

Thierry Vaira

BTS SN

v1.0 - 7 juillet 2017



Définition

- 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'algorithmie

- 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 (une erreur).
 - sélectionner un jeu de données en "entrées"
 - définir le résultat attendu en "sorties"
 - vérifier le résultat obtenu avec le résultat attendu

Bonnes pratiques

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 ! ».



Représentation

- Un algorithme peut être rédigé textuellement (en PSEUDO-CODE) ou graphiquement (algorithme, organigramme, logigramme, ...) :

```

Variable Nom_Utilisateur : Chaîne de
    caractères
Variable Nb_Lettres : Entier
Début
    Lire Nom_Utilisateur
    Nb_Lettres <- longueur(
        Nom_Utilisateur)
    Si Nb_Lettres > 10
        Ecrire "Votre nom est long"
    Sinon
        Ecrire "Votre nom est court"
    FinSi
Fin
  
```

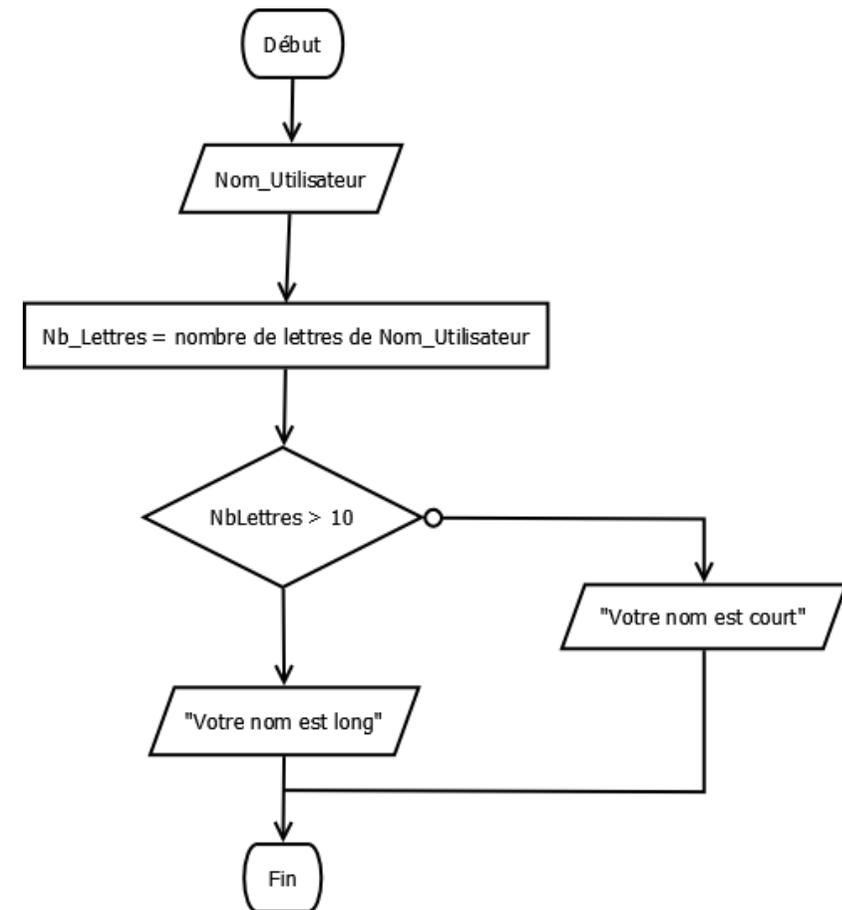


Figure: Norme ISO 5807

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 affecte une valeur à une variable
indice <- 0
```

Les variables dans les langages de programmation

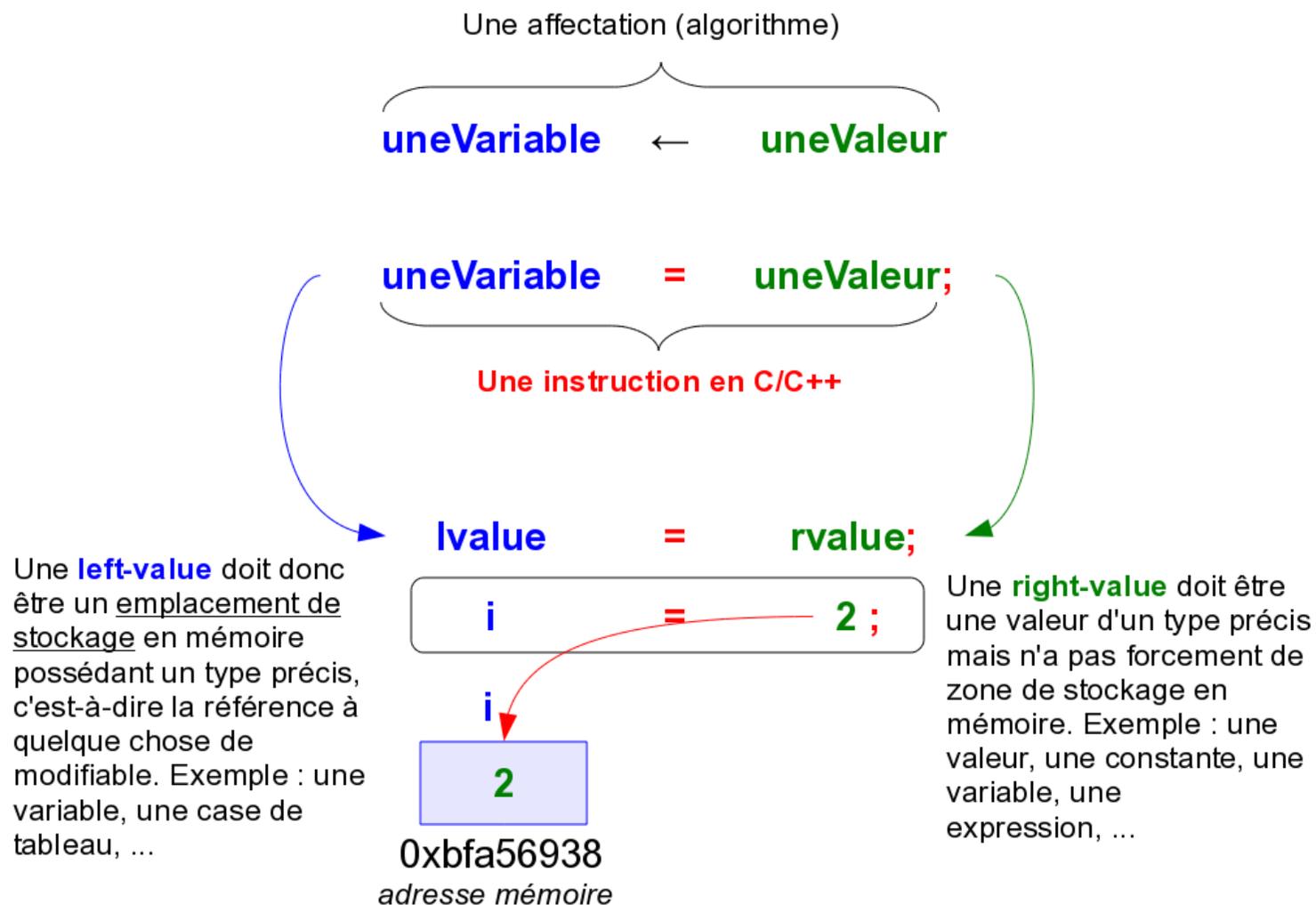
- Dans les langages compilés, les variables doivent être déclarées en précisant leur type. Il est conseillé de toujours les initialiser :
- Dans les langages interprétés, seules les valeurs ont un type ce qui n'oblige pas toujours à déclarer les variables :

```
// En C/C++ :  
  
// une variable entière  
int i = 0;  
// une variable réelle  
float j = 2.5;  
  
// interdit  
i = 2.5; // 2.5 n'est pas de  
         type entier
```

```
// En PHP :  
  
// une valeur entière  
$i = 0;  
// une valeur réelle  
$j = 2.5;  
  
// autorisée  
$i = 2.5; // la variable $i  
         n'a pas de type déclaré
```

L'affectation

- La première des instructions est la possibilité d'**affecter une valeur à une variable**. Dans un algorithme, on utilisera l'**opérateur d'affectation** \leftarrow :



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;
}
```

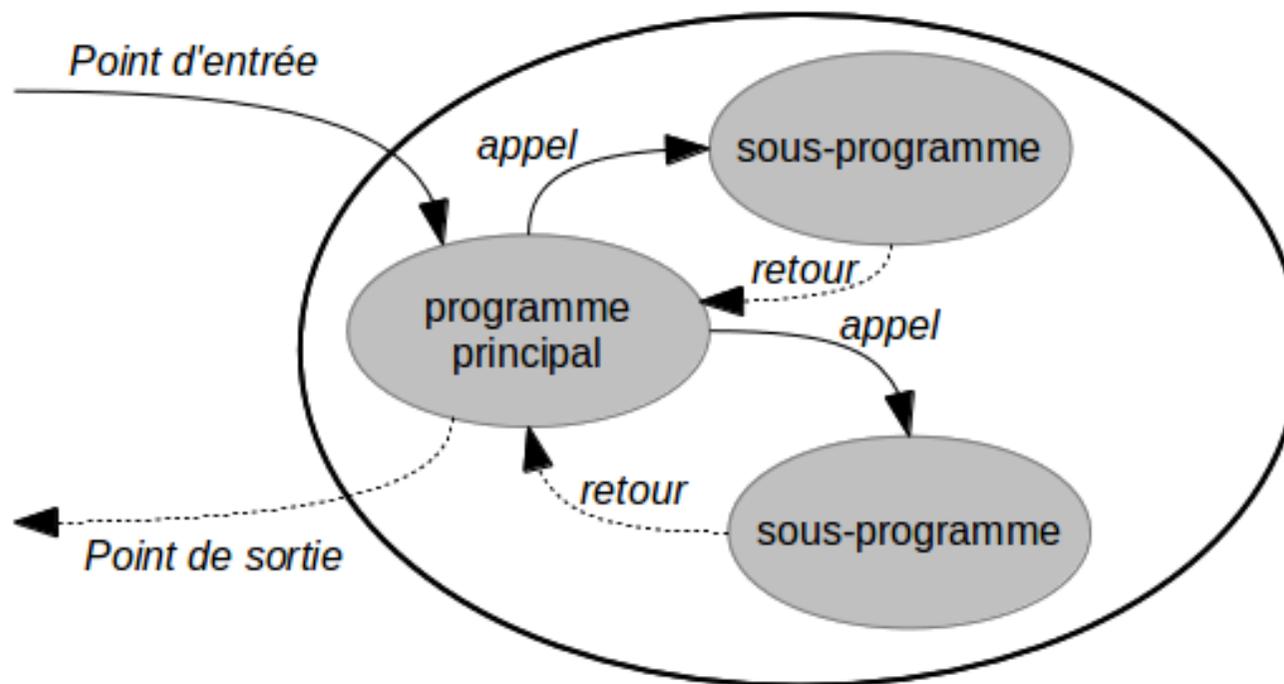
L'approche fonctionnelle

- 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

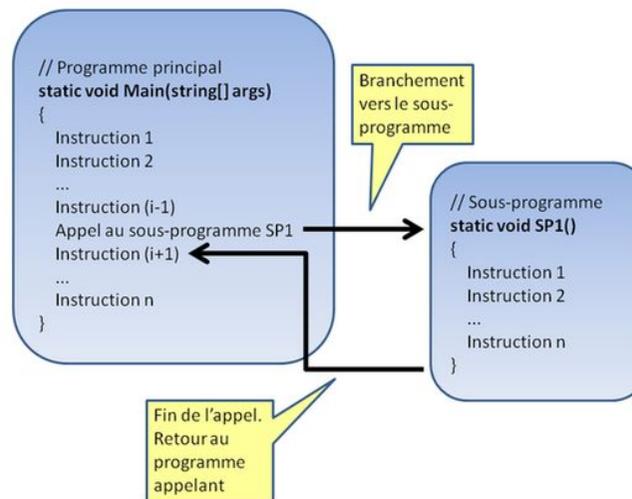
→ Dans le cas d'une approche fonctionnelle, un programme n'est plus une simple séquence d'instructions mais est constitué :

- 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 Development 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).

