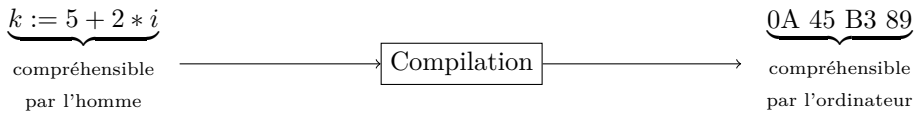
Chapitre 1

# À quoi ça sert ?



## 1.1 Qu'est ce que c'est ?

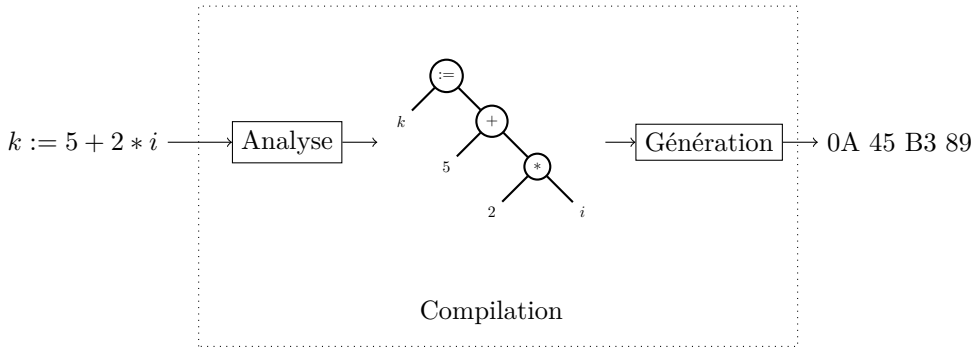
L'analyse lexicale et l'analyse syntaxique sont deux étapes du processus de *compilation*. La compilation est le procédé qui automatise la traduction depuis le langage haut niveau vers le langage bas niveau (les termes utilisés sont explicités ci-après).




Plus précisément, un ordinateur dispose d'un jeu d'opérations qu'il est capable de réaliser : opérations arithmétiques, accès mémoire, etc. Il reçoit une liste d'instructions écrite dans un langage spécifique à l'ordinateur : le langage machine. Ce langage est expressif mais il est « bas niveau » : on peut comprendre *localement* chaque instruction (additionner deux registres, écrire dans la mémoire, etc.) mais il est très difficile de saisir le sens *global* de ces instructions. À ce titre, le langage machine est difficilement compréhensible et il n'est pas adapté au programmeur. Ce dernier choisit de rédiger un programme dans un langage qu'il comprend et qui est adapté à ses besoins – un langage de « haut niveau ». Par exemple, un biologiste va choisir un langage qui décrit l'évolution d'une réaction biochimique ; un programmeur système va utiliser le langage C.

La compilation se décompose en deux phases :

1. La *phase d'analyse*, phase au cours de laquelle la machine essaye de comprendre le « sens » du programme. Concrètement, l'ordinateur transforme le texte du programme (représentation humaine) en une structure de données représentant le programme. Cette structure est un *arbre syntaxique*, communément appelée la « représentation abstraite ».
2. La *phase de génération*, phase au cours de laquelle la machine retranscrit le programme dans le langage cible choisi. Concrètement, l'ordinateur transforme l'arbre syntaxique en code machine.



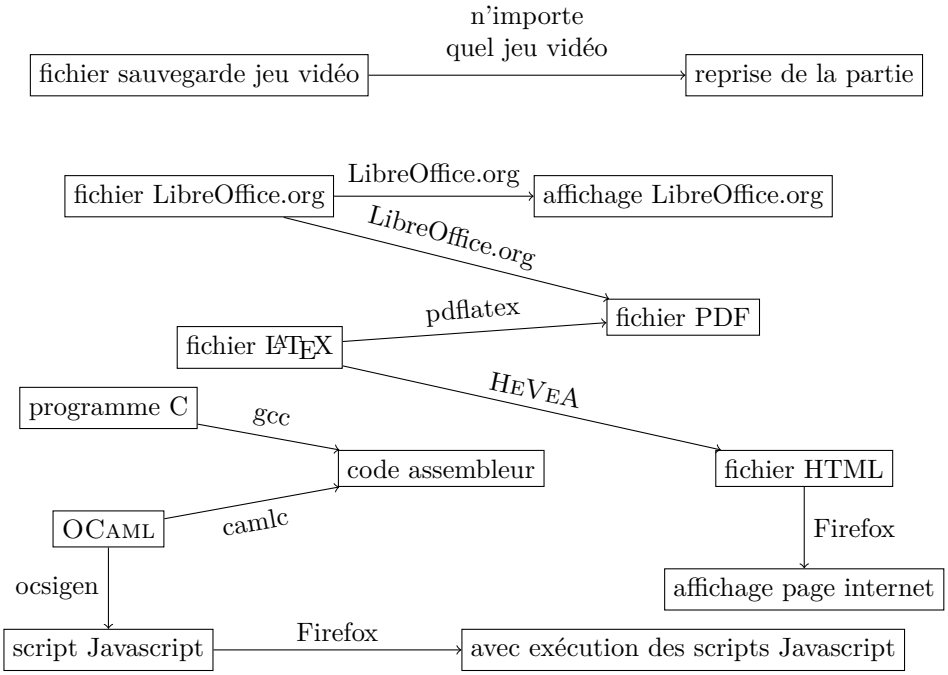

Dans le présent ouvrage, nous nous concentrons sur la phase d'analyse.

## 1.2 Les applications de la compilation

Les applications sont terriblement vastes :

- Certains compilateurs génèrent du code pour des machines dites virtuelles : par exemple Java ou OCaml.
- Certains compilateurs génèrent du code intermédiaire : citons Prolog qui produit du code dans le langage impératif C, ou le compilateur gcc qui produit du code intermédiaire *RTL* (« register transfer langage »).
- Le code généré n'est pas forcément un programme au sens habituel du terme : le compilateur LaTeX génère par exemple un document pdf.
- Transformer une expression régulière en un automate est un processus de compilation.
- Parfois la « représentation abstraite » peut être utilisée directement, on parle alors d'*interprétation*. Par exemple, une calculatrice ou un tableur réalisent une phase d'analyse sur une expression qui est ensuite calculée sans générer de code machine.
- Un navigateur internet réalise l'analyse d'un fichier html avant d'afficher la page correspondante.
- Aussi, certains logiciels (comme des traitements de texte, des logiciels de dessins, des jeux vidéos à sauvegarde, etc.) effectuent de même des analyses à l'ouverture de fichiers ; à l'enregistrement de fichiers, ils effectuent l'opération inverse : ils transforment le contenu de la mémoire en fichier texte.
- Les logiciels HeVeA et Latex2html sont des logiciels de conversion : ils transforment un fichier TeX en fichier html. Tout autre logiciel de conversion repose aussi sur les principes de la compilation (analyse et génération).

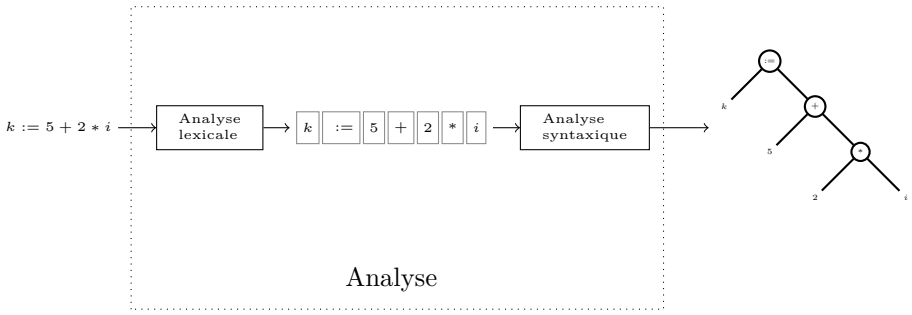
La compilation intervient partout dans la vie courante :




Lorsque vous lisez vos courriers électroniques, écrivez un document, surfez sur internet, utilisez votre calculatrice, jouez aux jeux vidéos et bien sûr lorsque vous programmez, la compilation est là ! Merci à la compilation qui nous a permis d'écrire ce livre.

## 1.3 Les deux étapes de la phase d'analyse

La phase d'analyse se décompose en deux étapes : l'étape d'analyse lexicale et l'étape d'analyse syntaxique.




L'analyse lexicale est un pré-traitement facilitant l'analyse syntaxique. Elle regroupe les lettres du texte en mots. Avant l'analyse lexicale, le texte est une suite de lettres ; après l'analyse lexicale, le texte est une suite de mots. L'analyse syntaxique réalise la construction de l'arbre syntaxique.

L'analyse lexicale et syntaxique sont abordées respectivement dans la partie II et la partie III.

## 1.4 En synthèse

En synthèse, on peut résumer ainsi :

1. l'analyse lexicale : regroupement des lettres en mots ;
2. l'analyse syntaxique : construction de l'arbre à partir des mots.

Une fois l'arbre construit, il est possible de réaliser n'importe quelle opération envisagée :

— génération de code pour un processeur,
— exécution directe de l'arbre (on parle *d'interprétation*),
— de réaliser une vérification du typage du programme,
— de conduire une analyse sémantique,
— ou toute opération souhaitée.

Comme évoqué précédemment, nous nous concentrons sur la phase d'analyse, c'est-à-dire sur des méthodes de construction de l'arbre.

Deuxième partie

# L'analyse lexicale

Chapitre 2

# L'analyse lexicale



L'analyse lexicale est le procédé qui regroupe les lettres en mots.

## 2.1 Quel est le problème ?

Rappelons le contexte. Le but de la phase d'analyse est d'extraire la structure (l'arbre syntaxique) à partir d'une description textuelle. Une description textuelle est une suite de lettres. Une description textuelle possède donc deux caractéristiques : elle est linéaire et elle est composée d'unités qui sont des caractères (des lettres ASCII). Par exemple, la phrase de Boris Vian « Je ne veux pas gagner ma vie, je l'ai. » (cf. [Via]) est la suite de lettres suivantes :

| J | e | ␣ | n | e | ␣ | v | e | u | x | ␣ | p | a | s | ␣ | g | a | g | n | e | r | ␣ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| m | a | ␣ | v | i | e | , | ␣ | j | e | ␣ | l | ' | a | i | · | | | | | | | | |

Le but final est d'obtenir la structure de la phrase (information non-linéaire *a priori*). L'opération qui consiste à reconstruire le caractère non-linéaire du programme à partir de sa description linéaire s'appelle *l'analyse syntaxique*.

Avant de procéder à la reconstruction de la structure, il est nécessaire d'isoler les mots de la phrase :

| Je | ne | veux | pas | gagner | ma | vie | , | je | l' | ai | · |
|---|---|---|---|---|---|---|---|---|---|---|---|

L'*analyse lexicale* est cette phase qui précède l'analyse syntaxique et qui consiste à transformer la suite de lettres en la suite de mots. Elle transforme une description linéaire en une description linéaire moins granuleuse et dont les éléments ont du sens pour la reconnaissance de la structure.

Les mots produits par l'analyse lexicale sont appelés *lexèmes* (en anglais, des « tokens »). En l'occurrence, l'analyse lexicale de « Je ne veux pas gagner ma vie, je l'ai. » produit 12 lexèmes.

**Remarque 1** .....
Ainsi, l'analyse syntaxique se réalise sur la liste de lexèmes. On pourrait imaginer une analyse syntaxique qui se réaliserait sur une liste de lettres. Mais l'analyse syntaxique serait trop compliquée :

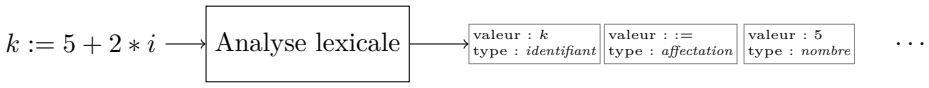
- L'analyse lexicale permet de s'abstraire de la mise en page. Par exemple, les textes « 1 + 2 » et « 1 + 2 » représentent les même programmes. L'analyse syntaxique n'aura pas à s'en préoccuper.
- L'analyse lexicale permet de réaliser une première filtration. Par exemple, les lexèmes de type *commentaire* sont utiles pour l'humain qui rédige ou qui lit le texte, mais ils sont inutiles pour la machine – ces lexèmes seront donc ignorés par la machine lors de l'analyse syntaxique.
- L'analyse lexicale repère déjà des lexèmes de différents types (nombre, opérateurs, mots de la langue française, symboles de ponctuation, etc.). Le type de lexèmes est une information essentielle pour l'analyse syntaxique.

**Remarque 2** .....
L'analyse lexicale est parfois utilisée seule. Citons l'exemple de la coloration du texte d'un programme qui aide le programmeur en mettant en valeur les lexèmes qui composent le programme. Par exemple, le programme PASCAL suivant a été mis en forme :

```
function factorielle(n : integer) : integer;
begin
    if n = 0 then factorielle := 1
    else factorielle := n * factorielle(n - 1)
end; { factorielle }
```

## 2.2 Types de lexèmes

Un lexème est composé de deux éléments : sa valeur et son type. L'analyse lexicale calcule à la fois la valeur et le type des lexèmes :




Les deux informations sont importantes :

- L'analyse syntaxique n'utilisera que le type du lexème. De plus, un analyseur syntaxique qui lit le lexème *nombre-naturel* puis lit le lexème *symbole-affectation* reconnaît un problème puisque l'opérateur d'affectation est nécessairement précédé d'une variable (remarquons que l'analyseur syntaxique n'a pas besoin de savoir de quel nombre il s'agit, il a juste besoin de savoir qu'il s'agit d'un nombre).
- La valeur sera utilisée lors de l'exécution du programme (faire $1+1$ ou $105+69$ donnent des résultats différents !).

## 2.3 Règles d'analyse lexicale

Pour reconnaître un lexème et déterminer son type, l'analyse lexicale réalise une opération de « *pattern matching* » (en français, une « reconnaissance de motif »). L'analyse lexicale recherche dans le texte un lexème qu'elle reconnaît. Par exemple, en lisant le texte « 12 », l'analyse lexicale reconnaît le lexème « 12 » de type nombre.

En lisant le texte « myvar », l'analyse lexicale reconnaît le lexème « myvar » de type identifiant. Pour réaliser cette opération de reconnaissance, l'analyse lexicale utilise des *expressions régulières* (voir encadré).

**Les expressions régulières**

Les expressions régulières sont une notation pour représenter des ensembles de mots. À cette fin, elles ont recours aux opérateurs suivants :

| | |
|---|---|
| $\mid$ | ou |
| $*$ | zéro, une ou plusieurs itérations |
| $+$ | une ou plusieurs itérations |

Par exemple, l'expression régulière « 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 » est une notation qui représentent l'ensemble des chiffres : $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Pour raccourcir l'expression régulière, on note « 0 | 1 | $\cdots$ | 9 ». Nous avons une expressions régulière qui décrit les chiffres ; pour obtenir les nombres, il suffit d'adjoindre l'opérateur + : $(0 \mid 1 \mid \cdots \mid 9)+$.

Le mot qui ne contient aucune lettre est appelé le « mot vide » et il est noté $\varepsilon$.

Un ensemble de mots est appelé un « langage ». Attention ! Cette appellation peut être contre-intuitive : l'intuition nous dit qu'un langage est un ensemble de phrases bien construites, alors qu'ici il ne s'agit que d'un ensemble de mots (définition moins restrictive).

Voici quelques exemples d'expressions régulières :

| | |
|---|---|
| $\varepsilon$ | le langage contenant le mot vide : $\{\varepsilon\}$ |
| $1^+$ | les suites non vides de 1 : $\{1, 11, 111, 1111, \dots\}$ |
| $1^*$ | les suites éventuellement vides de 1 : $\{\varepsilon, 1, 11, 111, 1111, \dots\}$ |
| $(0 \mid 1 \mid \cdots \mid 9)^+$ | les suites non vides de chiffres : $\{12, 52, 92517, \dots\}$ |

L'expression régulière « un (très)$^*$ (bon)$^+$ gâteau » représentent les phrases qui commencent par « un », puis une suite (éventuellement vide) de « très », puis une suite non vide de « bon » puis « gâteau ». On dit que l'expression régulière *reconnaît* les mots suivants :

- un bon gâteau ;
- un très bon bon bon gâteau ;
- un très très très très très bon gâteau ;
- etc.

Le lecteur est invité à lire des ouvrages de références comme [Car08] et [Sak].

**Définition 3** (règle d'analyse lexicale)

Soit $A$ l'alphabet des caractères.
Soit $T$ l'ensemble des types de lexème.
Une *règle d'analyse lexicale* est la donnée :

- d'une expression régulière $e$ sur $A$, dont le langage associé ne contient pas le mot vide $\varepsilon$,
- d'un type de lexème $t \in T$.

Cette règle est notée $e \to t$.