# Partie I : Vue d'ensemble et bases

- Pourquoi étudier la construction de compilateurs ?
- Le rôle et la structure d'un compilateur.
- Langage et syntaxe.
- Langages formels.



# Pourquoi étudier la construction de compilateurs ?

Très peu de gens écrivent des compilateurs comme profession.

Alors pourquoi apprendre à construire des compilateurs ?

- Un informaticien compétent comprend les langages de haut niveau ainsi que le matériel.
- Un compilateur relie les deux.
- C'est pourquoi comprendre les techniques de compilation est essentiel pour comprendre comment les langages de programmation et les ordinateurs interagissent.
- Beaucoup d'applications contiennent de petits langages pour leur configuration et pour flexibiliser leur contrôle.
  - Exemples : les macros de Word, les scripts pour le graphisme et l'animation, les descriptions de structures de données.

## Pourquoi étudier la construction de compilateurs ? (suite)

- Les techniques de compilation sont nécessaires pour correctement implanter ces langages d'extension.
- Les formats de données sont aussi des langages formels. De plus en plus de données en format interchangeable ressemblent à un texte d'un langage formel (p.ex. HTML, XML).
- Les techniques de compilation sont utiles pour lire, manipuler et écrire des données.
- Mis à part cela, les compilateurs sont d'excellents exemples de grands systèmes complexes
  - qui peuvent être spécifiés rigoureusement,
  - qui peuvent être réalisés seulement en combinant théorie et pratique.



## Le rôle d'un compilateur

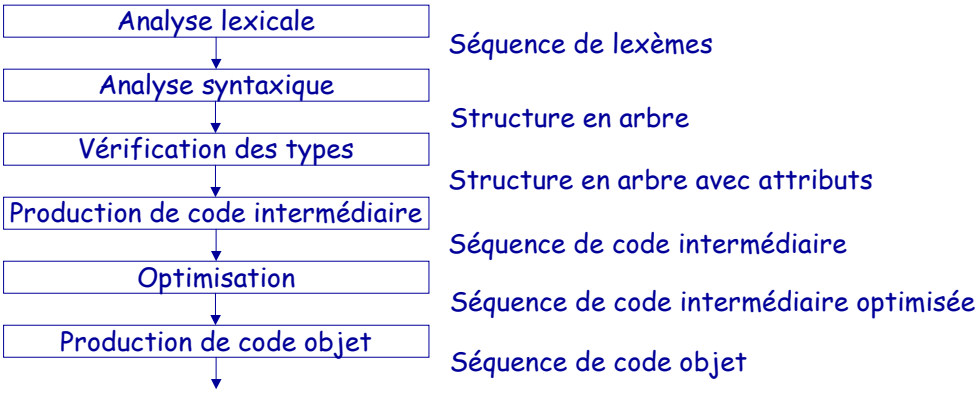
- Le rôle principal d'un compilateur est de traduire des programmes écrits dans un langage *source* donné en un langage *objet*.
- Souvent, le langage source est un langage de programmation et le langage objet est un langage machine.
- Quelques exceptions : traduction source-source, traduction de code machine, manipulation de données en XML.
- Une partie du travail du compilateur est aussi de détecter si un programme donné est conforme aux règles du langage source.
- Une spécification d'un compilateur est constituée par
  - une spécification de son langage source et de son langage objet,
  - une spécification du processus de traduction de l'un en l'autre.

## Langages

- Formellement, un langage est un ensemble de *chaînes de caractères* que l'on appelle les *phrases* du langage.
- En pratique, chaque phrase possède une *structure* qui peut être décrite par un *arbre.*
- Les règles régissant la structure des phrases sont définies par une *grammaire*.
- Exemples :
  - Les phrases d'un langage de programmation sont des programmes (légaux).
  - Un programme est une phrase constituée de mots (ou : *symboles, lexèmes*); sa structure est donnée par une grammaire.
  - Un mot est lui-même une séquence de caractères dont la structure peut aussi être donnée par une grammaire.



## Structure d'un compilateur




- Les phases ne sont pas nécessairement exécutées l'une après l'autre.
- Les structures de données intermédiaires n'existent parfois jamais dans leur intégralité.

## Langage et syntaxe

- Les phrases d'un langage ont une structure déterminée par une grammaire.
- **Exemple**: Une phrase correcte est formée par un sujet suivi d'un verbe.
- Ceci peut être exprimé par une grammaire :

  Phrase = Sujet Verbe
- Complétons cela avec deux nouvelles *productions* :

  Sujet = "Pierre" | "Claire"

  Verbe = "cours" | "marche"
- Ceci définit 4 phrases possibles :

  Pierre cours | Pierre marche | Claire cours | Claire marche
- Généralement, les langages contiennent un nombre infini de phrases.



## Langage et syntaxe (suite)

- Un nombre infini de phrases peuvent être exprimées par un nombre fini de productions en définissant certains symboles récursivement.
- Exemple :

  Nombre = Chiffre | Chiffre Nombre

  Chiffre = "0" | "1" | "2" | "3" | "4" | . . . | "9".

Génère :

0
12
123
1024

etc.

## Langages formels

Une grammaire est formellement définie par :

- Un ensemble de *symboles terminaux.*
- Un ensemble de *symboles non-terminaux*
- Un ensemble de *règles syntaxiques* (ou : *productions*)
- Un *symbole initial.*

Une grammaire définit un langage formé par l'ensemble des séquences finies de symboles terminaux qui peuvent être dérivées du symbole initial par des applications successives des productions.



## Le langage des grammaires (non contextuelles)

```
grammar     = production grammar  |  (empty)
production  = identifier "=" expression "."
expression  = term  |  expression "|" term
term        = factor  |  term factor  |  "(empty)"
factor      = identifier  |  string
identifier  = letter  |  identifier letter  |  identifier digit
string      = "\"" stringchars "\""
stringchars = stringchars stringchar  |  (empty)
stringchar  = escapechar  |  plainchar
escapechar  = "\\" char
plainchar   = charNoQuoteNoEscape
char        = «any printable character».
charNoQuoteNoEscape = «any printable character except `"' and `\'.»
```

## Le langage des grammaires (non contextuelles) (suite)

- Il a été développé à l'origine par J. Backus et P. Naur pour la définition d'Algol 60.
- C'est pourquoi on l'appelle la *notation Backus-Naur* ou *BNF* (de l'anglais *Backus-Naur form*).
- Exercice : Déterminez le symbole initial, et les symboles terminaux et non-terminaux de cette grammaire.



## Notation Backus-Naur étendue

Les grammaires peuvent souvent être simplifiées et raccourcies en utilisant deux constructions supplémentaires :

- {x} exprime la *répétition :* zéro, une ou plusieurs occurrences de x.
- [x] exprime l'*option :* zéro ou une occurrence de x.

Ce nouveau formalisme est appelé *notation Backus-Naur étendue* ou *EBNF* (de l'anglais *extended Backus-Naur form*). Sa syntaxe est la suivante:

# Notation Backus-Naur étendue (suite)

```
syntax      =  {production}
production  =  identifier "=" expression "."
expression  =  term {"|" term}
term        =  {factor}
factor      =  identifier
            |  string
            |  "(" expression ")"
            |  "[" expression "]"
            |  "{" expression "}"
identifier  =  letter { letter  |  digit }
string      =  "\"" {stringchar} "\"
```

(le reste est identique au BNF)

Exercice : Écrivez la grammaire des nombres entiers (éventuellement signés) en BNF puis EBNF.