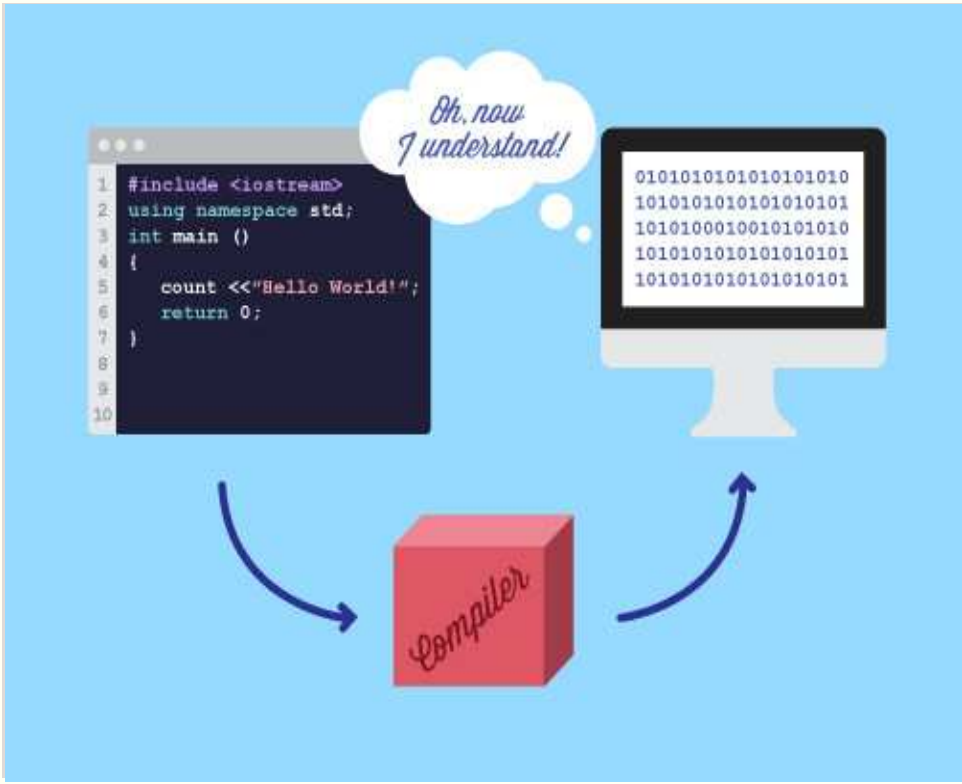
TECHNIQUES DE COMPILATION

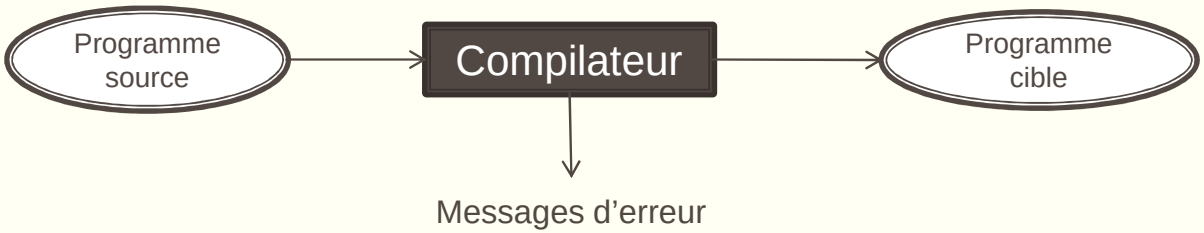


2ème SI
Houda Benali

ISSAT Mateur 2020/2021

Objectif du cours

- Comprendre ***les principes*** de construction d'un ***Compilateur*** (***les techniques et les outils***)



- ***Programme***: description d'une ***suite d'opérations*** qui étant donné une entrée va produire un résultat
- ***Les opérations*** sont décrites dans plusieurs modèles de calcul (impératif, fonctionnel, objet, assembleur . . .)
- ***Langages sources*** : Fortran, Pascal, C, Cobol, SQL., Maple …
- ***Langages cibles*** : Langage machine, Code intermédiaire, Autre langage de programmation

Pourquoi étudier les techniques de compilation ?

- **Une réalité**: Très peu de gens écrivent des compilateurs comme profession !!

  ➔ *On n'étudie pas les techniques de compilation pour seulement construire des compilateurs !*

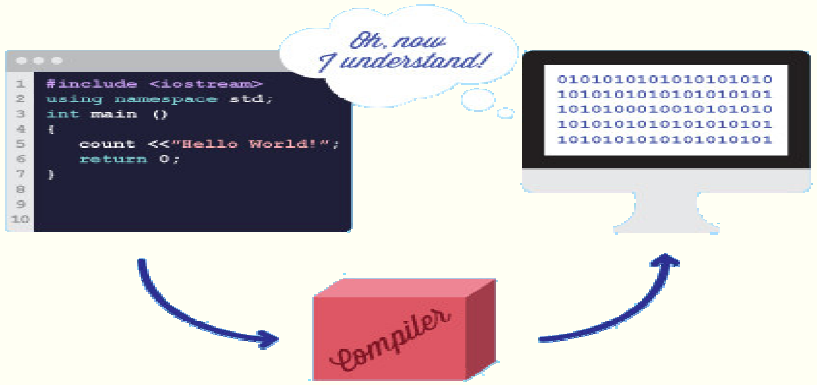
- **De vraies raisons :**
  - Çà vous rendra plus ***compétent***: Un compilateur est un logiciel complexe dont la production est relativement bien maîtrisée.
  - Beaucoup d'applications ***contiennent de petits langages*** pour leur configuration et pour flexibiliser leur contrôle :
    - les macros de Word, les scripts pour le graphisme et l'animation, ...
  - Les techniques de compilation sont nécessaires pour correctement implanter des ***langages d'extension***.
  - Les programmes lisent des données et dans les interfaces actuelles on propose ***un format souple***
    - De nombreux modules logiciels permettent de transcrire un ***texte résultant d'une saisie*** en une ***représentation interne*** sur laquelle on applique les traitements qui font le « cœur » de l'application.

PLAN DU COURS

1. Compilateur : définition, modèle et concepts
2. Théorie des langages : notions de base
3. Analyse Lexicale
4. Analyse Syntaxique
5. Analyse Sémantique
6. Production de code

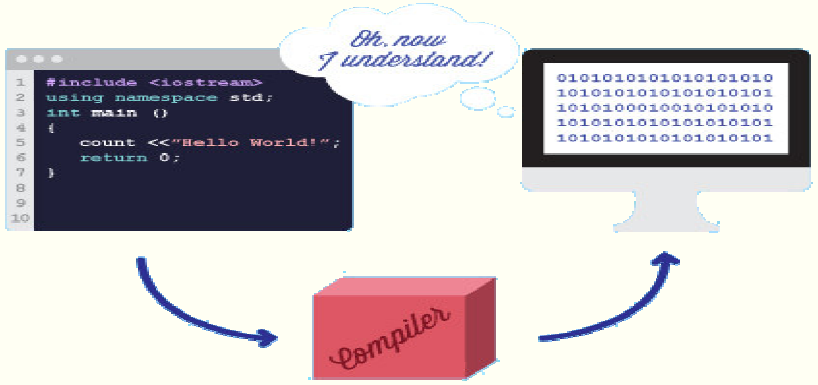
COMPILATEUR : DÉFINITION, MODÈLE ET CONCEPTS

C’est quoi un Compilateur ?



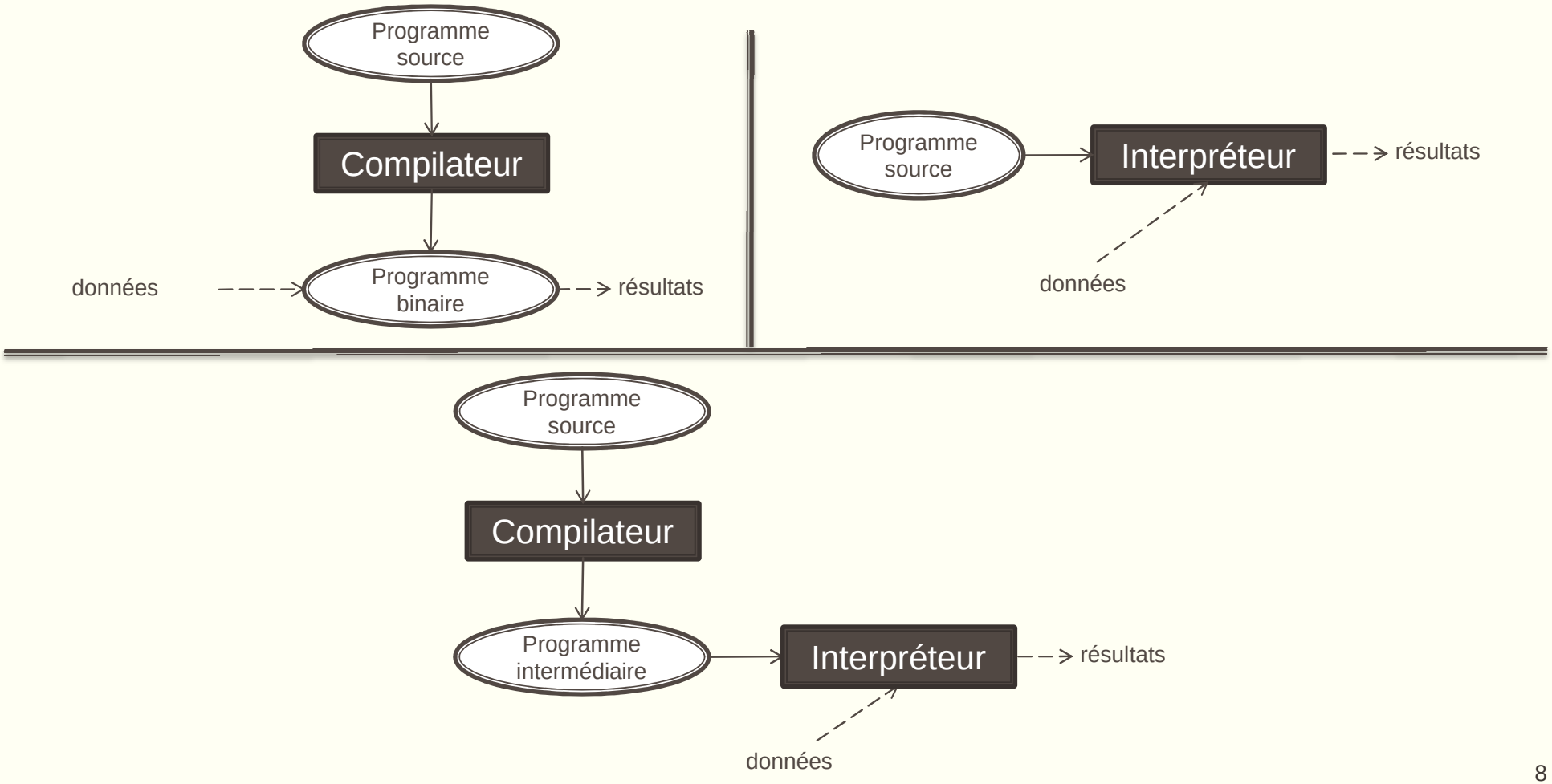
- Un compilateur est un ***traducteur*** qui permet de transformer un programme écrit dans un langage L1 en un autre programme écrit dans un langage machine L2.
- En pratique on s’arrête souvent à ***un langage intermédiaire*** (assembleur ou encore langage d’une machine abstraite).

C’est quoi un Compilateur ?



- Le compilateur peut rejeter des programmes qu’il considère incorrects, dans le cas contraire, il construit un nouveau programme (phase statique) que la machine pourra exécuter sur différentes entrées.
- L’exécution du nouveau programme sur une entrée particulière peut ne pas terminer ou échouer à produire un résultat (phase dynamique).

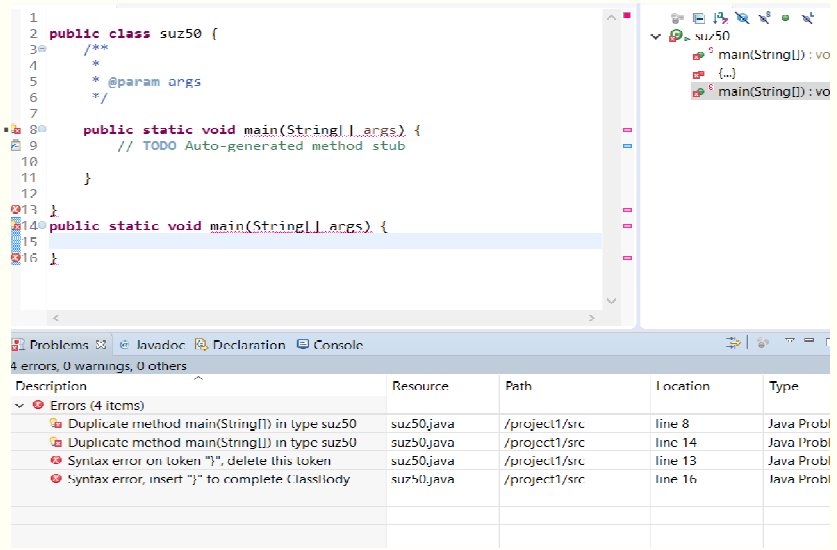
Compilateur vs. Interpréteur



Qu’attend-on d’un compilateur ?

1. Détection des erreurs

- Identificateurs mal formés, commentaires non fermés . .
- Constructions syntaxiques incorrectes
- Identificateurs non déclarés
- Expressions mal typées
- . . .



- **Remarque !**
  - Les erreurs détectées à la compilation s’appellent les erreurs **statiques.**
  - Les erreurs détectées à l’exécution s’appellent les erreurs **dynamiques**: division par zéro, dépassement des bornes dans un tableau. .

Qu’attend-on d’un compilateur ?

2. Exactitude

- Le programme compilé doit représenter le même calcul que le programme original.

3. Efficacité

- Le compilateur doit être si possible rapide (en particulier ne pas boucler)
- Le compilateur doit produire un code qui s’exécutera aussi rapidement que possible

Phases de la compilation

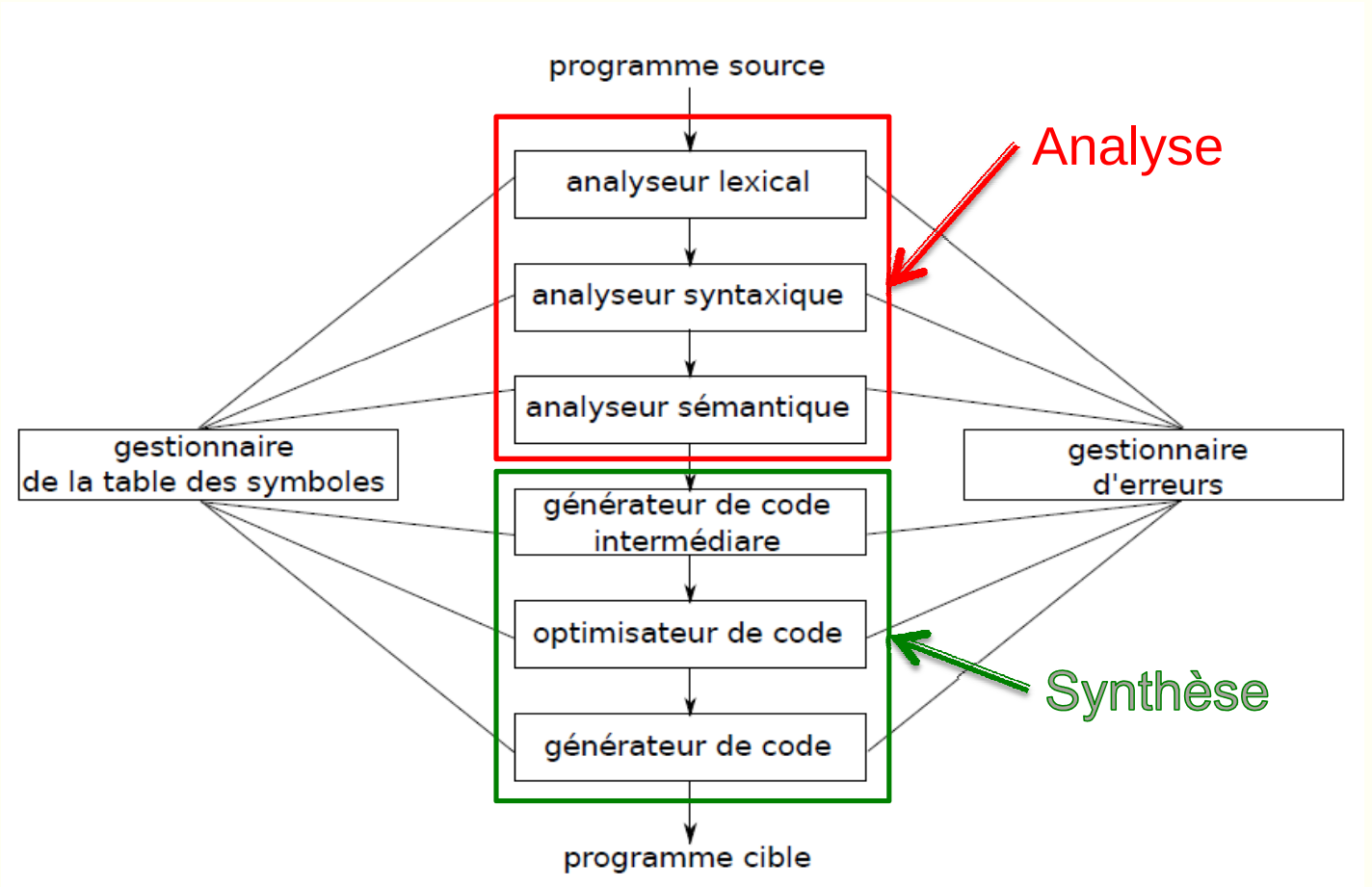


Table des symboles

- Initialisée par l'analyseur lexical, complétée et utilisée par les autres phases.
- Permet de stocker des informations sur les variables et les fonctions du programmes.
  - Stockage des variables
    - Le nom
    - Le type
    - La portée (globale ou locales)
    - L'emplacement mémoire
  - Stockages des fonctions
    - Le nom
    - La portée (globale ou locales)
    - Nom, type et mode de passage des arguments
    - Type de la valeur retournée

Gestion des erreurs

- Erreurs ***lexicales*** (caractères interdits).
- Erreurs ***syntaxiques*** (règles structurelles non respectées).
- Erreurs ***sémantiques*** (incohérence des opérations : contrôle statique).

⇒Diverses récupérations possibles, selon la finalité du compilateur :

- arrêt à la première erreur ;
- resynchronisation sur la prochaine construction correcte ;
- tentatives de correction.

L’analyse du programme source

- Découpage en trois phases :
  1. Analyse lexicale : flot de caractères regroupés en **unités lexicales**
  2. Analyse syntaxique : regroupement des unités lexicales en **unités grammaticales**
  3. Analyse sémantique : contrôle ou établissement de la **cohérence sémantique**

Analyse Lexicale

- Découpe le programme source en éléments appelés ***lexèmes***
- Peut faire des ajouts dans la table des symboles
- L'analyseur lexical transmet à l'analyseur syntaxique les lexèmes sous forme de ***couples*** (type de lexème, valeur du lexème)
  - exemple (nombre, 123)
- Les symboles terminaux (ou type de lexème) constitue l'interface entre l'analyseur lexical et l'analyseur syntaxique.

Analyse Syntaxique (1/3)

- Transformer une suite de caractères en un ***arbre de syntaxe abstrait*** représentant la description des opérations à effectuer.
- Le programme source vérifie un certain nombre de ***contraintes syntaxiques***.
- L'ensemble des contraintes est appelé ***grammaire*** du langage source.
- Si le programme ne respecte pas la grammaire du langage, il est considéré incorrect et le processus de compilation est échoué.

Analyse Syntaxique (2/3)

- Les contraintes syntaxiques sont représentées sous la forme de ***règles de réécriture***;
  - La règle A → BC indique que le symbole A peut se réécrire comme la suite des deux symboles B et C ;
- L'ensemble des règles de réécriture constitue la ***grammaire du langage***
- La grammaire d'un langage L permet de générer ***tous*** les programmes corrects écrits en L ***et seulement ceux ci***

Analyse Syntaxique (3/3)

- Dans la règle A → ß
  - A est appelé partie gauche de la règle
  - *∂* est appelé partie droite de la règle
- Lorsque plusieurs règles partagent la même partie gauche :
  - A → ß1; A → ß2 ; ... ; A → ßn
  - on les note A → ß1| ß2 |…| ßn

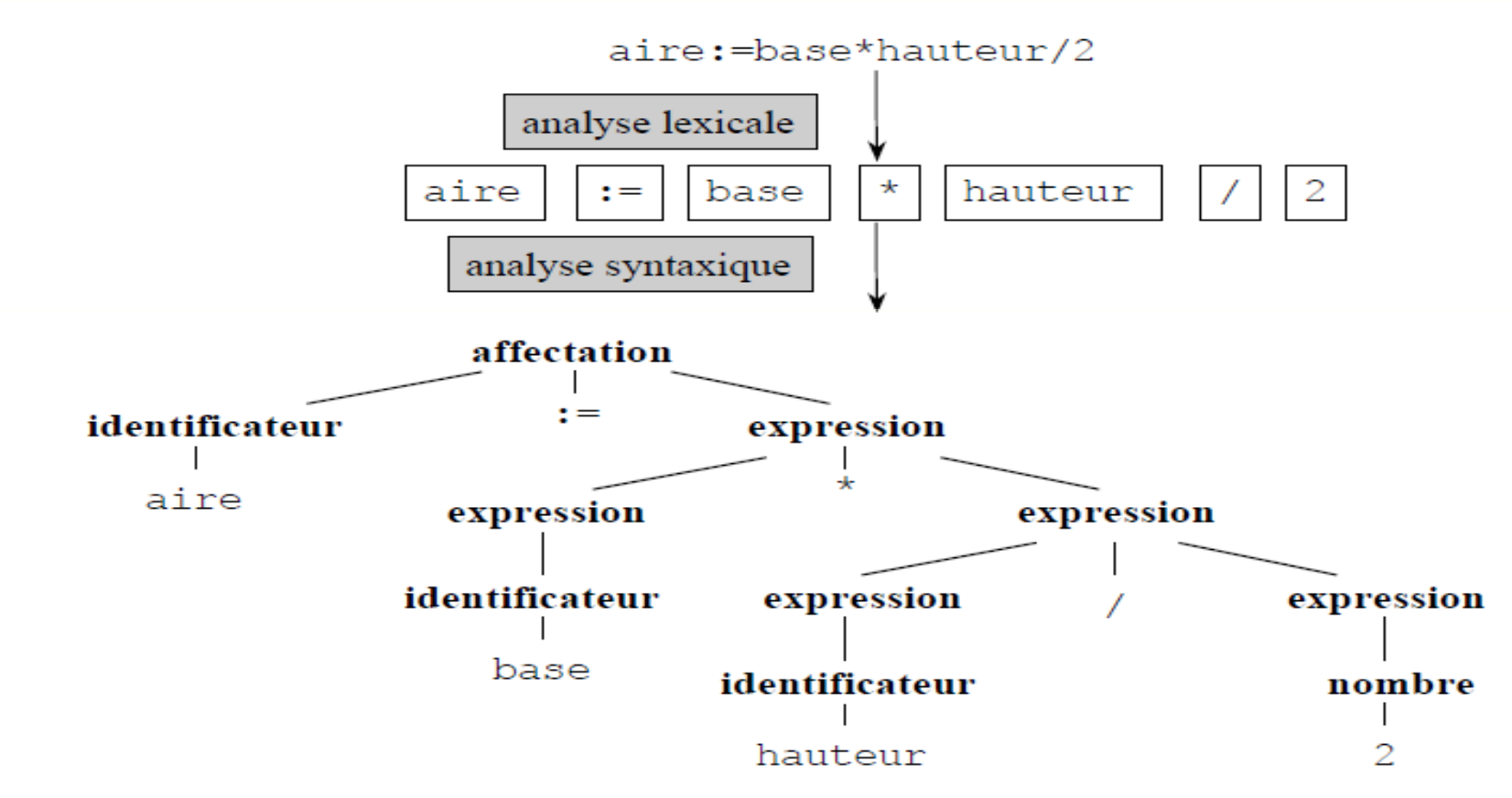
Exemple de grammaire d'une expression arithmétique

```
E       -> E + E
         | E - E
         | E * E
         | E / E
         | ( E )
         | NOMBRE
NOMBRE  -> CHIFFRE
         | CHIFFRE NOMBRE
CHIFFRE -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

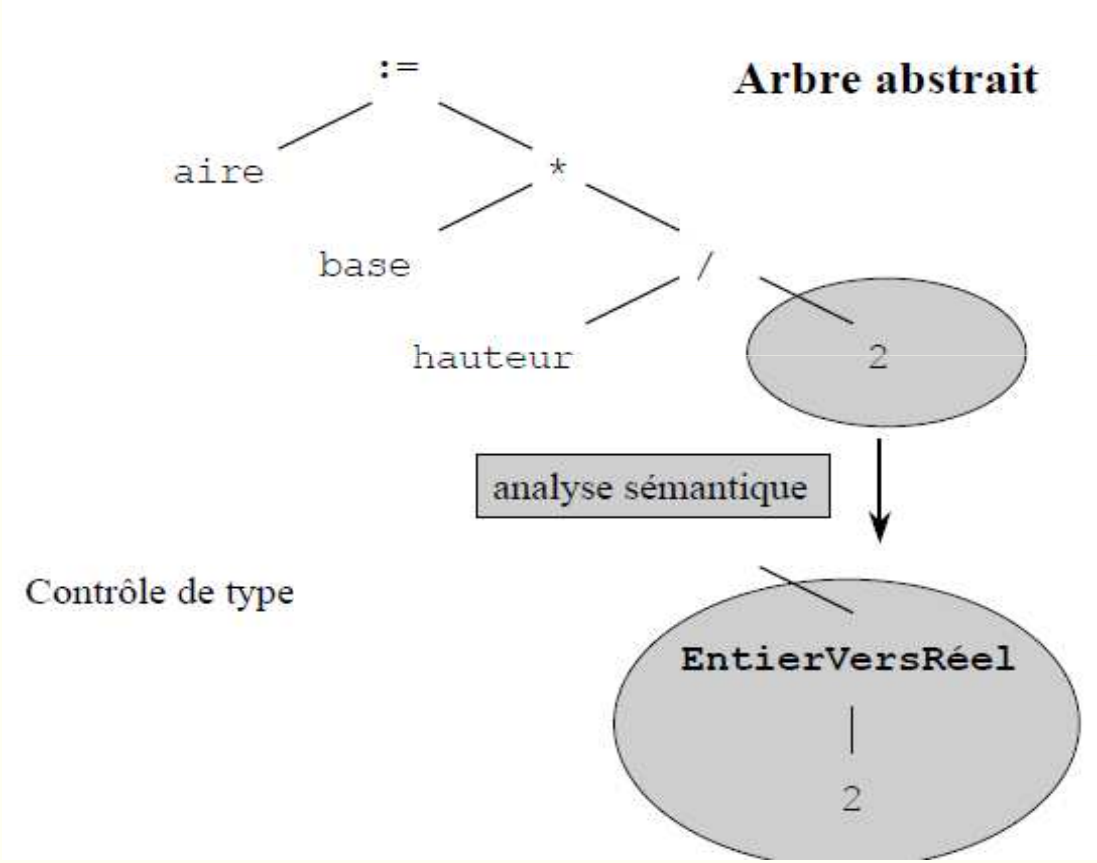
Analyse Sémantique

- L'analyse sémantique utilise l'***arbre abstrait***, ainsi que la ***table de symboles*** afin d'effectuer un certain nombre de contrôles sémantiques, parmi lesquels :
  - Vérifier que les variables utilisées ont été bien déclarées.
  - Le contrôle de type : le compilateur vérifie que les opérandes d'un operateur possèdent bien le bon type.
  - Conversions automatiques de types.

L’analyse du programme source : Exemple



L’analyse du programme source : Exemple



Phase de Synthèse

- Passage par plusieurs langages intermédiaires pour produire un code efficace
- La production de code est effectuée lors du parcours de la Représentation Intermédiaire (RI) construite à l'issue de l'étape d'analyse (Arbre syntaxique abstrait).