كلـيـــــــــة العلوم
السملالية - مراكش
FACULTE DES SCIENCES
SEMLALIA-MARRAKECH

Département d'Informatique

LF-SMI S6

Compilation

Pr. Aimad QAZDAR

aimad.qazdar@uca.ac.ma

14 avril 2021

Pourquoi étudier la compilation ?

- C'est une des branches les plus dynamiques de l'informatique, et une des plus anciennes à mériter ce qualificatif.

- Son domaine d'application est très large : conversion de formats ; traduction de langages ; optimisation de programmes.

- Elle permet de mettre en pratique de nombreux algorithmes généraux.

Plan

Ch. 1 : Introduction Générale

Ch. 2 : Analyse Lexicale

Ch. 3 : Analyse Syntaxique

Ch. 4 : Analyse Sémantique

Ch. 5 : Génération de code

Bibliographie

- Un livre standard, appelé « **dragon book** »
  - Compilateurs : Principes, techniques et outils. A. Aho, R. Sethi et J. Ullman. InterEditions.

- Autres livres intéressants :
  - Compilateurs. D. Grune, H.E. Bal, G. J.H. Jacobs, K.G. Langendoen. Editions Dunod.

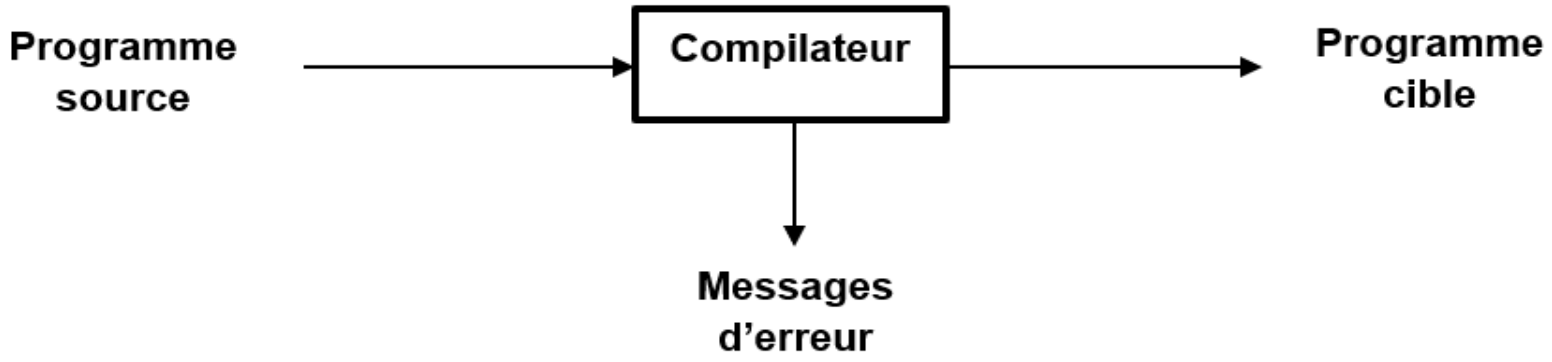
Chapitre 1 :

Introduction générale

La compilation et ses enjeux...

Qu’est ce qu’la compilation?

- C’une **transformation** que l'on fait subir à un **programme** écrit dans un **langage évolue** pour le rendre **exécutable** (*sens usuel du terme*).
- C'est une traduction : un **texte** écrit en C, C++, Java, etc., exprime un **algorithme** et il s'agit de **produire** un **autre texte**, spécifiant le **même algorithme** dans le **langage** d'une **machine** que nous cherchons à programmer.



Les niveaux de langages

- L’objectif de l’informaticien est de définir un langage qui:
  - assure une communication efficace avec la machine
  - facilite la description des besoins d’un programmeur.
- Les langages sont passés par différents niveaux :
  - **Le langage machine** : c’est jeu d’instruction de l’UC. {0,1} communication sans traduction
  - **Le langage d’assemblage** : utilise des noms mnémoniques pour le code opérations, les opérandes et les adresses de données.
  - **Le langage de haut niveau** : notation plus agréable et plus souple permettant l’utilisation de structures de données et de contrôles complexes.
  - **Le langage de commande** : permet de communiquer avec un système d’exploitation.

Les niveaux de langage

- **Le langage machine**
  - Ensemble d ’instructions codées en hexadécimal, permettant de manipuler les éléments du microprocesseur et de la mémoire. Propre à chaque microprocesseur.

```
00111110101110
00111001111101
1111110111011
```

- **Le langage assembleur**
  - Traduction mot à mot du langage machine en mnémoniques Apparitions des identificateurs de données et des étiquettes de branchement

```
mov ax,500  ;
mov bx,0x20 ;
add ax,bx   ;
```

- **Le langage haut niveau**
  - Notation plus agréable et plus souple permettant l’utilisation de structures de données et de contrôles complexes
  - Premiers langages de haut niveaux (Fortran, Algol, Pascal, C, etc.)

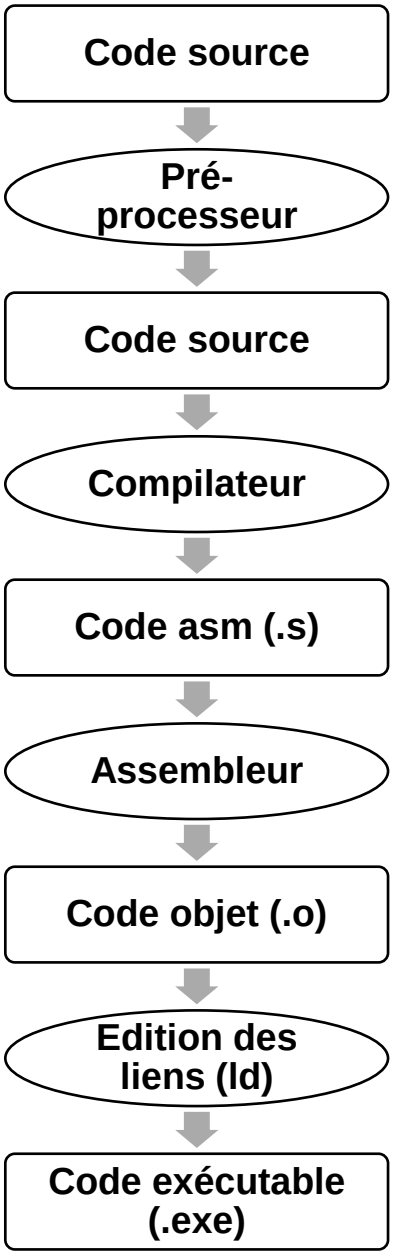
```
int x = 500;
int y = 32;
x+=y;
```

- **Le langage de commande**
  - Permet de communiquer avec un système d’exploitation.

```
[root@centos home]# cd /tmp
[root@centos tmp]# pwd
/tmp
[root@centos tmp]# mkdir myDir
```

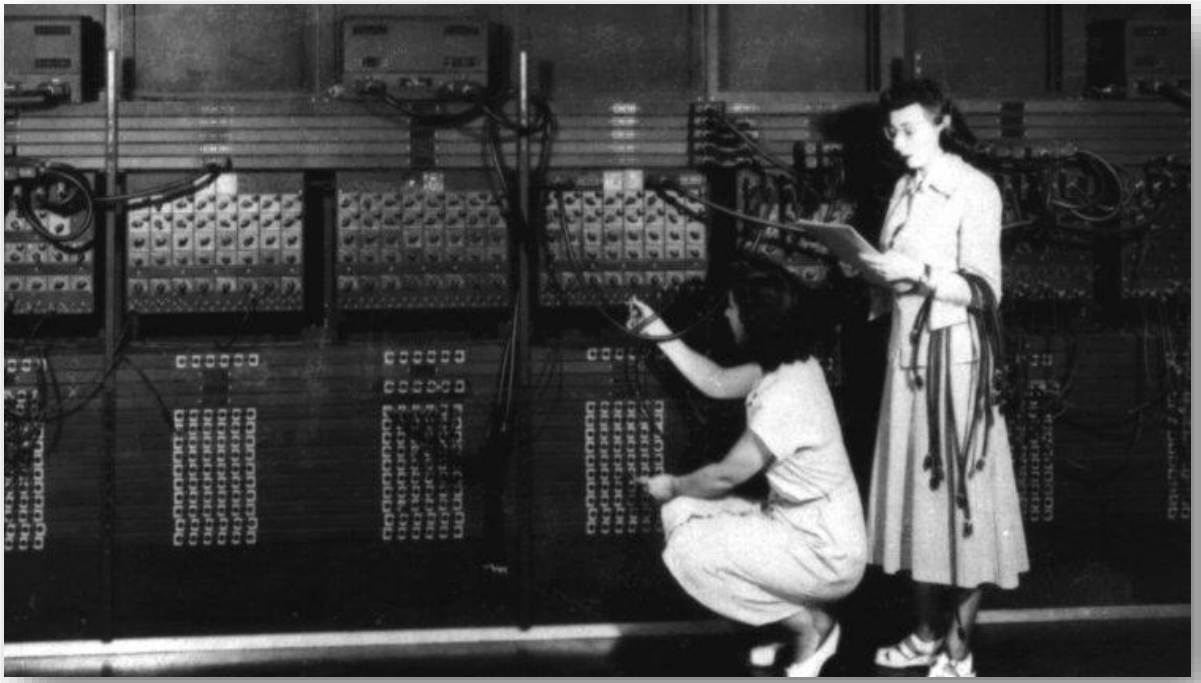
Chaine de transformation d'un programme

- En pratique, un compilateur fait toutes ces étapes présentées dans cette figure.
- **Assembleur** :
  - Transforme des instructions asm (texte) en binaire. C'est une correspondance presque directe.
  - Résolution de certaines étiquettes.
  - Suit les directives (déclarations, macros, etc.)
- **Édition des liens (link)** :
  - Remplacer les noms des variables et fonctions par des adresses.
  - Fusionner les .o pour créer une application complète (cas d'un link statique)
  - Générer les appels vers le chargeur dynamique (cas d'un link dynamique)



Histoire des compilateurs

Dans les années 1940, les premiers ordinateurs sont construits



L'ENIAC (Electronic Numerical Integrator And Computer) à l'Université de Pennsylvanie

Histoire des compilateurs

- L'**ENIAC** :
  - Sert à calculer des tables de tir de calculs balistiques
  - Remplace 200 personnes qui calculaient manuellement les tables
  - Personne n'avait jamais programmé un ordinateur
  - Il fallait programmer de façon manuelle, en enfichant des câbles et en manipulant des commutateurs
- **Aucune documentation à part les schémas techniques du câblage de la machine**

Histoire des compilateurs

- Les premières **programmeuses** étaient 6 femmes :

Kathleen Antonelli

Jean Jennings Bartik (à gacuhe)
Frances Bilas Spence (à droite)

Frances Synder Holberton

Marlyn Wescoff Meltzer

Ruth Lichterman Teitelbaum

Histoire des compilateurs

- Au début des années 1950, Grace Murray Hopper est programmeuse sur le Harvard Mark I
- Elle lance l'idée « folle » de créer un nouveau langage de programmation, basé sur l'anglais, pour indiquer à l'ordinateur ce qu'il doit faire
- Personne ne la prend au sérieux, un ordinateur est seulement fait pour faire du calcul, et rien d'autre!

Hopper, Grace Murray
working on the Bureau of Ordnance Computation Project at Harvard University, 1946.
U.S. Department of Defense

Histoire des compilateurs

- Elle écrit le premier programme qui interprète du code et écrit le code machine associé.
- C’est ce qu’on appelle aujourd’hui **compiler un programme**!
- Elle crée ensuite le langage COBOL

Grace Hopper en 1984
(*1906 – 1992*)

Histoire des compilateurs

- **De 1945 à 1960**
  - On faisait de la génération de code
  - Comment produire du code pour une machine spécifique?
  - Souvent de l’assembleur au code machine manuellement
  - Les programmes compilés étaient souvent plus longs et moins efficaces que ceux écrits par des programmeurs

Histoire des compilateurs

- **Développement des premiers langages**
  - FORTRAN (FORmula TRANslator) – 1954
  - LISP (LISt Processing) – 1958
  - ALGOL (ALGorithmic Oriented Language) – 1958
  - COBOL (Common Business Oriented Language) – 1959

- **ALGOL** introduit deux concepts novateurs :
  - Structure en blocs imbriqués
  - Notions de portée

Histoire des compilateurs

- **De 1960 à 1975**
  - Introduction de l'analyse syntaxique et des formalismes
  - Prolifération des langages
  - Développement de la plupart des paradigmes retrouvés dans les langages modernes
    - Concept de classe
    - Ramasse-miette
    - Compilation JIT (Just-in-time)
  - La majorité des langages sont basés sur les langages précédents

Histoire des compilateurs

- Inspiré d'ALGOL
  - SIMULA 1 (SIMple Universal LAnguage) – 1962
  - SIMULA 67 – 1967 : Ajout d'un modèle de classe et de la simulation par événement discret
  - C (évolution du langage B) – 1969-1973
  - SmallTalk – 1970
- Inspiré de FORTRAN
  - BASIC (Beginner's All-purpose Symbolic Instruction Code) – 1964
- ML (Meta Language) – 1973
  - Fournira l'inspiration pour Haskell et CamL

Histoire des compilateurs

- **De 1975 à 1990**
  - Consolidation des langages et des paradigmes
  - Développement d'une nouvelle architecture matérielle
    - Le matériel doit faciliter l'écriture des compilateurs, et non faciliter la vie des programmeurs assembleur!
  - Le problème devient : Comment optimiser le code?
    - Création de C++, Ada, Eiffel, Perl, etc.
    - Mettre l'emphase sur « comment compiler » au lieu de « vers quoi compiler »

Histoire des compilateurs

- **1990 et +**
  - Gestion du parallélisme, hiérarchie de mémoires
  - Apparition des langages « modernes »
  - Python – 1991
  - Java – 1995
  - Php et Javascript – 1995
  - C# – 2000
  - Etc.

Histoire des compilateurs

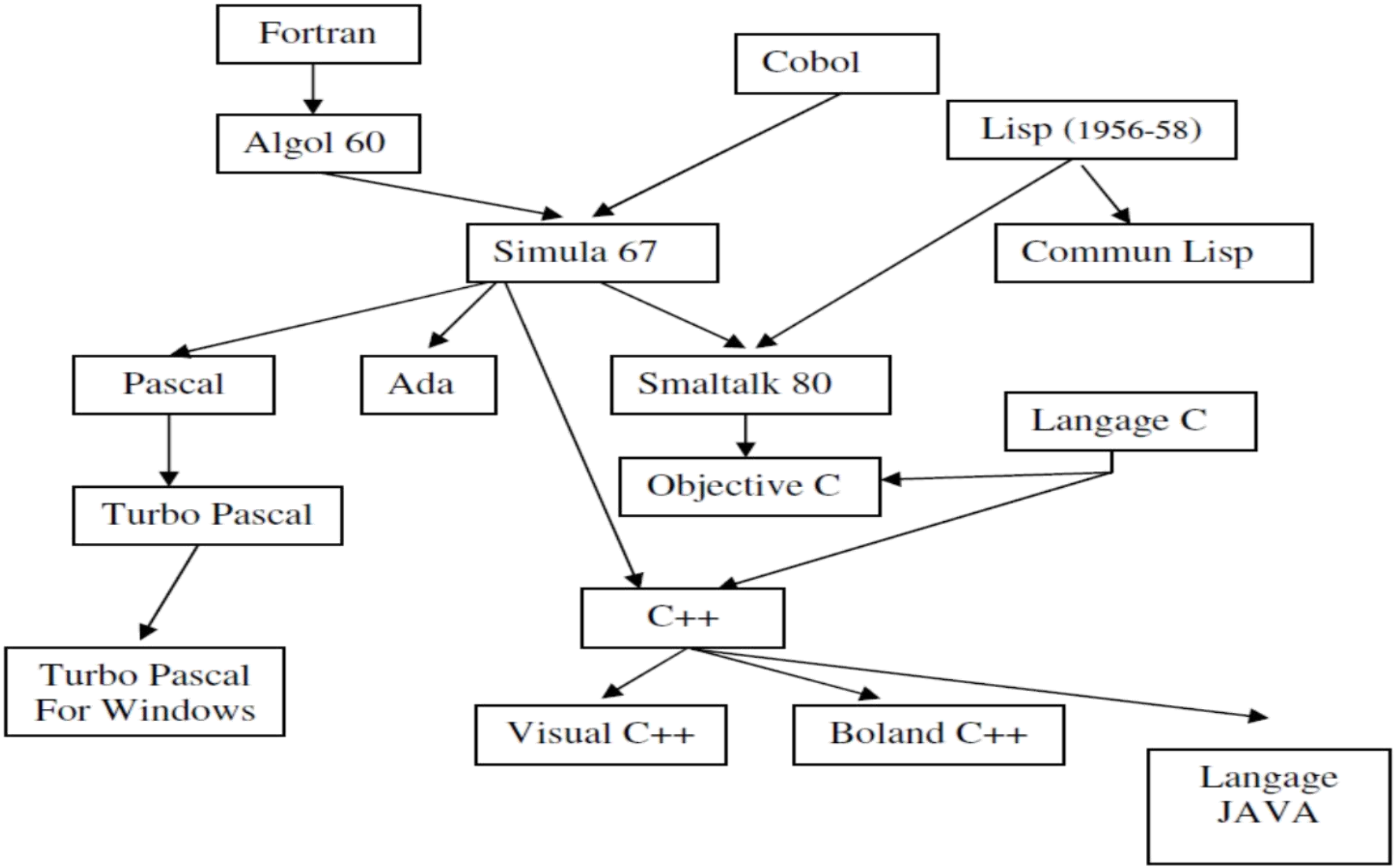
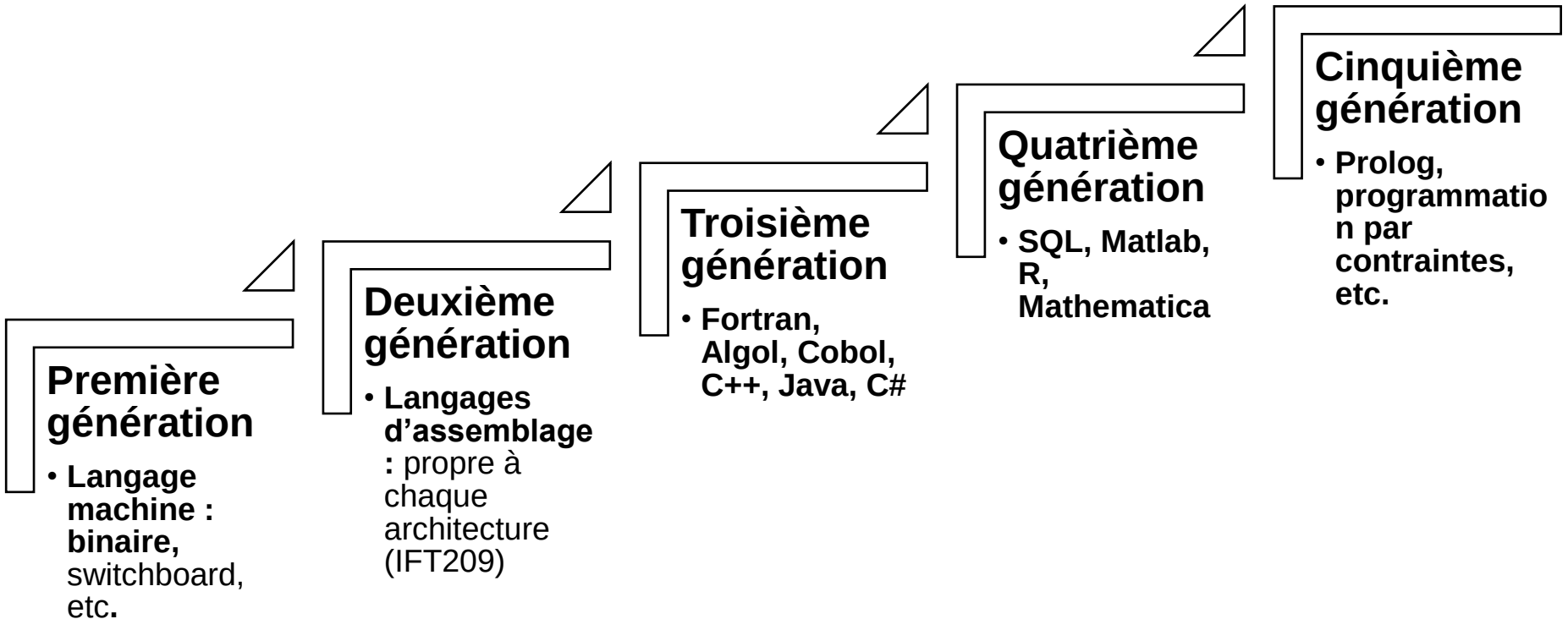


Schéma de l'évolution des compilateurs

Génération des langages



Quelques qualités d'un compilateur

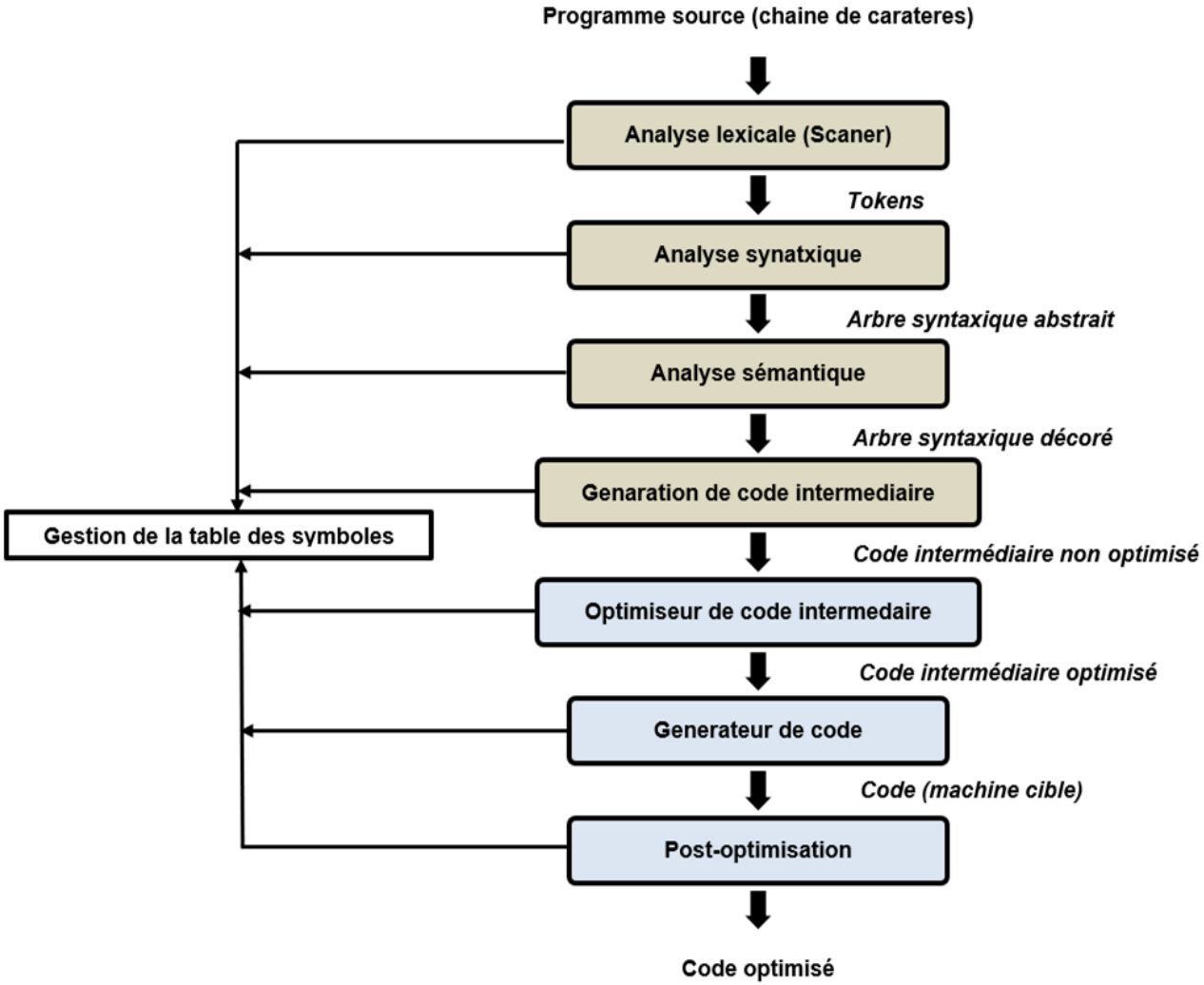
- **Confiance**
  - S'approcher du « zéro bug »
- **Optimisation**
  - Code généré rapidement
- **Efficacité**
  - Compilateur rapide
- **Economie**
  - Compilateur portable
- **Productivité**
  - Erreurs signalées utiles

Terminologie

- **Lexicalement correcte ?**
  - Oui papA
  - int i, j k void ; main(]; 54j
- **Syntaxiquement correcte ?**
  - Oui Moi demain partir
  - I = i j;
- **Sémantiquement correcte ?**
  - Le ciel éplucha l'oiseau.
  - int i; i = i + 1.5;

Phases d'un compilateur

Cette structure existe dans tous les compilateurs.



Le code natif est soit du **asm**, soit du **.o**, soit du **binaire**.

Phases d’un compilateur

- **Analyse lexicale** :
  - Dans cette phase, les caractères isolés qui constituent le texte source sont regroupés pour former des unités lexicales, qui sont les mots du langage.
  - L'analyse lexicale opère sous le contrôle de l'analyse syntaxique ; elle apparait comme une sorte de fonction de « lecture améliorée » , qui fournit un mot lors de chaque appel.

- **Analyse syntaxique :**
  - L'analyse lexicale reconnait les mots du langage, l'analyse syntaxique en reconnait les phrases.
  - Le rôle principal de cette phase est de dire si le texte source appartient au langage considère, c'est-a-dire s'il est correct relativement à la grammaire de ce dernier.

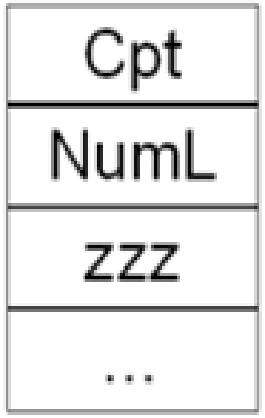
- **Analyse sémantique**
  - La structure du texte source étant correcte, il s'agit ici de vérifier certaines propriétés sémantiques, c'est-à-dire relatives à la signification de la phrase et de ses constituants :
    - les identificateurs apparaissant dans les expressions ont-ils été déclarés ?
    - les opérandes ont-ils les types requis par les opérateurs ?
    - les opérandes sont-ils compatibles ? n'y a-t-il pas des conversions a insérer ?
    - les arguments des appels de fonctions ont-ils le nombre et le type requis ?

- **Génération de code intermédiaire**
  - Après les phases d'analyse, certains compilateurs ne produisent pas directement le code attendu en sortie, mais une représentation intermédiaire, une sorte de code pour une machine abstraite.
  - Cela permet de concevoir indépendamment les premières phases du compilateur (constituant ce que l'on appelle sa face avant) qui ne dépendent que du langage source compile et les dernières phases (formant sa face arrière) qui ne dépendent que du langage cible ; l'idéal serait d'avoir plusieurs faces avant et plusieurs faces arrière qu'on pourrait assembler librement3.

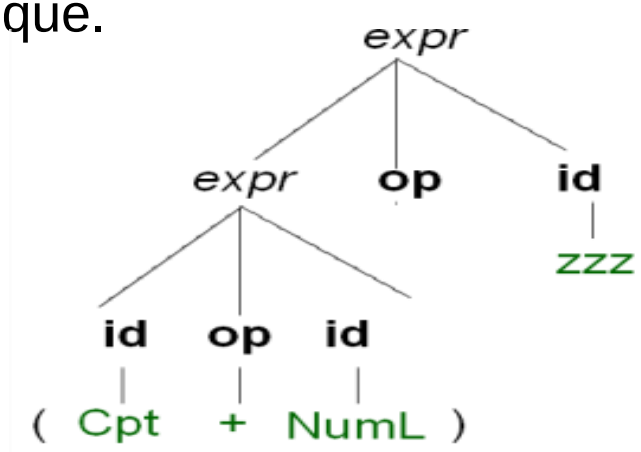
Phase d’un compilateur

- **Optimisation du code**
  - Il s'agit généralement ici de transformer le code afin que le programme résultant s’exécute plus rapidement. Par exemple
    - détecter l'inutilité de recalculer des expressions dont la valeur est déjà connue,
    - transporter a l'extérieur des boucles des expressions et sous-expressions dont les opérandes ont la même valeur µa toutes les itérations
    - détecter, et supprimer, les expressions inutiles.
    - Etc.
- **Génération du code final**
  - Cette phase, la plus impressionnante pour le néophyte, n'est pas forcement la plus difficile à réaliser.
  - Elle nécessite la connaissance de la machine cible (réelle, virtuelle ou abstraite), et notamment de ses possibilités en matière de registres, piles, etc.

- **Table des symboles** :
  - Dictionnaire contenant des informations sur les entités du programme à compiler (variables, fonctions, types, etc.).
  - Initialisée dès la phase d'analyse lexicale;
  - Mise à jour continuelle au fur et à mesure.



- **Arbre syntaxique** (arbre de dérivation):
  - Décrit la structure syntaxique d'un programme.
  - Construit par l'analyseur syntaxique.
  - Décoré par l'analyse sémantique.

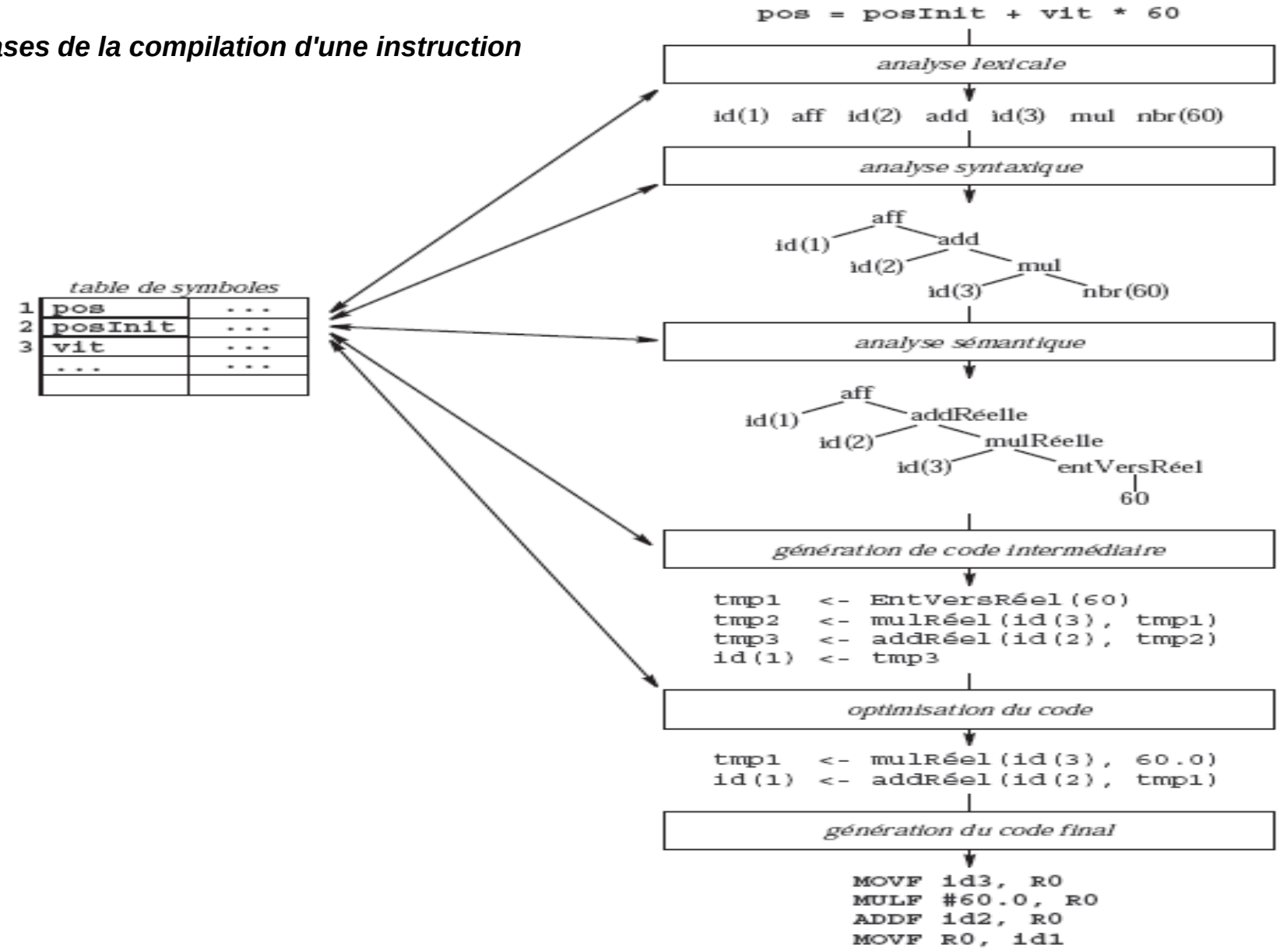


Code cible

- Un compilateur peut générer :
  - Un code assembleur cible
  - Un code d'un autre langage de programmation haut niveau (ex: C vers fortran, C++ vers java)
  - Représentation intermédiaire (ex: java vers bytecode)

Exemple

Phases de la compilation d'une instruction



Processeurs

- Processeurs à hautes performances (stations de travail, PC, super calculateurs, etc.)
  - Alpha (HP), UltraSparc (Sun), Pentium (Intel), Itanium (HP/Intel), Power (IBM), ...
- Processeurs embarqués : ils sont partout...
  - Mips, PowerPC, ARM, Xscale, ST 2xx
- Microcontrôleurs:
  - Électroménager, voiture, robots, etc.
- On peut les classer selon d'autres critères

Jeu d'instructions (ISA)

- C'est l'ensemble des instructions exécutées par un processeur.
- Un des critères de classification :
  - CISC (Complex Instruction Set Computer)
    - Intel est dominant (x86). AMD.
  - RISC (Reduced Instruction Set Computer)
    - majorité des autres processeurs.
- L'ISA est un langage de programmation bas niveau :
  - Non portable, difficile à vérifier/déboguer;
  - Programmes efficaces.
- Consulter le module ARCHITECTURE DES ORDINATEURS (S4)

Conclusion

- La compilation est une branche passionnante et ardue de l'informatique.
- Le **front-end** d'un compilateur permet:
  - Vérifier la validité lexico-syntaxico-sémantique du programme vis à vis de son langage.
  - Le traduire en un langage intermédiaire
- Le **back-end** permet de générer un code bas niveau vers une architecture donnée.
- Le **cœur** du compilateur analyse et optimise le programme