ÉCOLE CENTRALE D'INFORMATIQUE
UNIVERSITÉ CENTRALE
HONORIS UNITED UNIVERSITIES

CHAPITRE 1- INTRODUCTION À LA COMPILATION

Mme Fatma KAABI

Qu’est-ce qu’un compilateur ?

- Un compilateur est un programme qui a comme entrée un code source écrit en langage de haut niveau (langage évolué) est produit comme sortie un code cible en langage de bas niveau (langage d’assemblage ou langage machine).

- La traduction ne peut être effectué que si le code source est correct car, s’il y a des erreurs, le rôle du compilateur se limitera à produire en sortie des messages d’erreurs

Qu’est-ce qu’un compilateur ?

▪ Un compilateur est un traducteur qui permet de transformer un programme écrit dans un langage L_1 en un autre programme écrit dans un langage machine L_2.

Qu’est-ce qu’un programme ?

▪Description de comment à partir d’une entrée, produire un résultat.
▪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 programme sur une entrée particulière peut ne pas terminer ou échouer à produire un résultat (phase dynamique).

Qu'attend-on d'un compilateur ?

- Détection des erreurs :
  - Identificateurs mal formés, commentaires non fermés . . .
  - Constructions syntaxiques incorrectes
  - Identificateurs non déclarés
  - Expressions mal typées *if 3 then "toto" else 4.5*
  - Références non instanciées

  . . .

- 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 ?

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

Correction

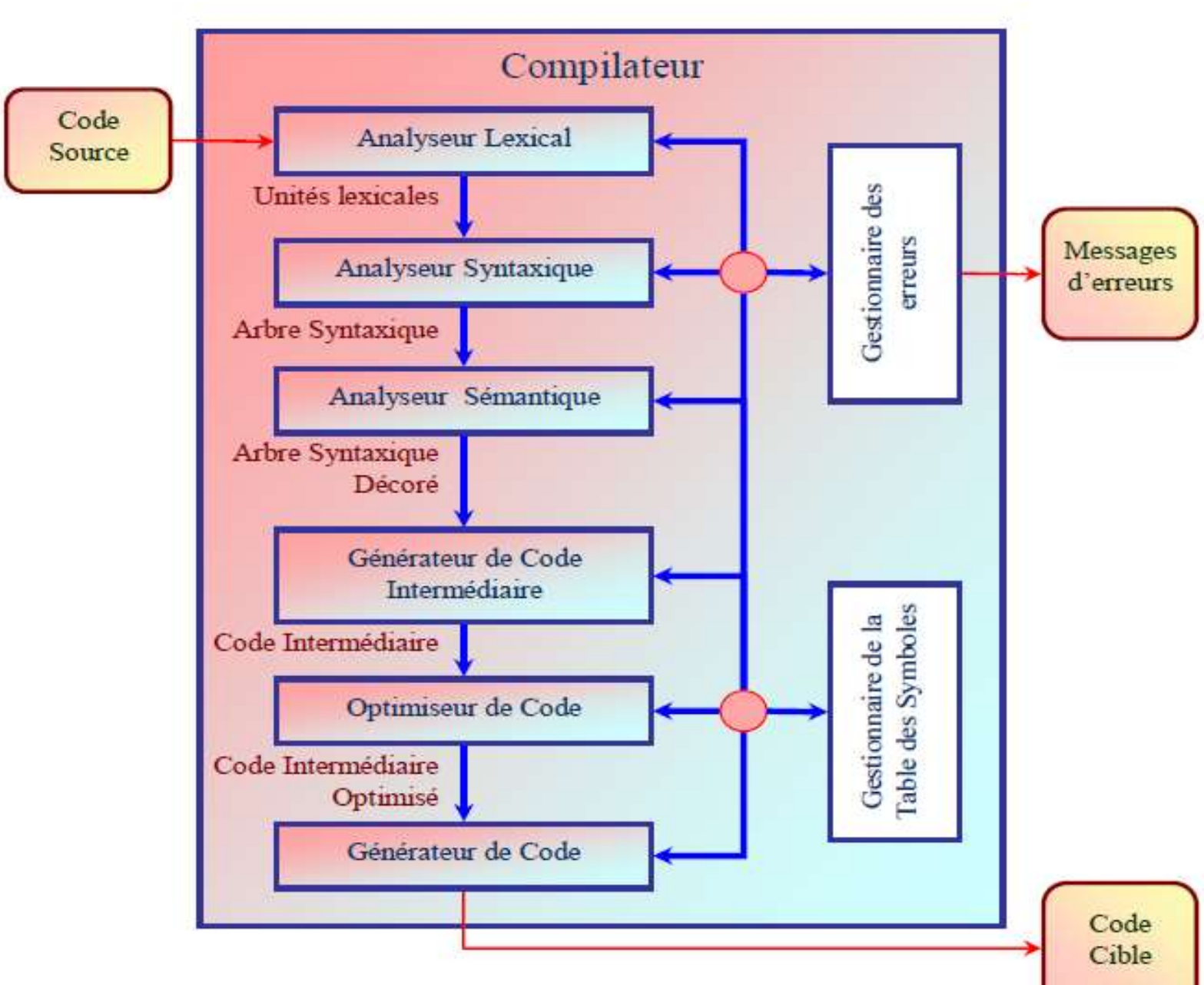
- Le programme compilé doit représenter le même calcul que le programme original.
- Nécessite d’avoir une description indépendante des calculs représentés par les programmes du langage source.

Structure d’un compilateur

- Premier compilateur : compilateur Fortran de J. Backus (1957)

- Un compilateur est généralement composé de modules correspondant aux phases logiques de l’opération de compilation

Structure d'un compilateur



L’analyseur lexical

- Connu aussi sous l’appellation Scanner**,** l’analyseur lexical a pour rôle principal la lecture du texte du code source (suite de caractères) puis la formation des unités lexicales (appelées aussi entités lexicales, lexèmes, jetons, tokens ou encore atomes lexicaux).

- Les principales sortes d'unités lexicales qu'on trouve dans les langages de programmation courants sont :

  - les caractères spéciaux simples : +, =, etc. ;
  - les caractères spéciaux doubles : <=, ++, etc. ;
  - les mots-clés : if, while, etc. ;
  - les constantes littérales : 123, -5, etc. ;
  - et les identificateurs : i, vitesse_du_vent, etc.

Exemple

Considérons l'expression d'affectation **a := b + 2 * c ;**

Les unités lexicales qui apparaissent dans cette expression sont :

Unité lexicale	Sa nature
a	Identificateur de variable
:=	Symbole d'affectation
b	Identificateur de variable
+	Opérateur d'addition
2	Valeur entière
*	Opérateur de multiplication
c	Identificateur de variable
;	Séparateur

Rôles de l’analyseur lexical

- Lire une suite de caractères de l’entrée et produire une suite de jetons (adéquats pour l’analyse syntaxique).
- Débarrasser le programme des commentaires et des espacements.
- Tenir à jour les coordonnées (ligne, colonne) dans le programme pour fin de production des éventuels messages d’erreur.
- . . .

Gestion des erreurs

- Plusieurs erreurs ne peuvent pas être géré au niveau de l'analyseur lexical. Par exemple, on retrouve dans le programme source la chaîne: fi (a == f(x)) . . .

- Cette erreur nous semble de nature lexicale à cause de la faute d'orthographe mais 'fi' constitue un identificateur tout à fait valide

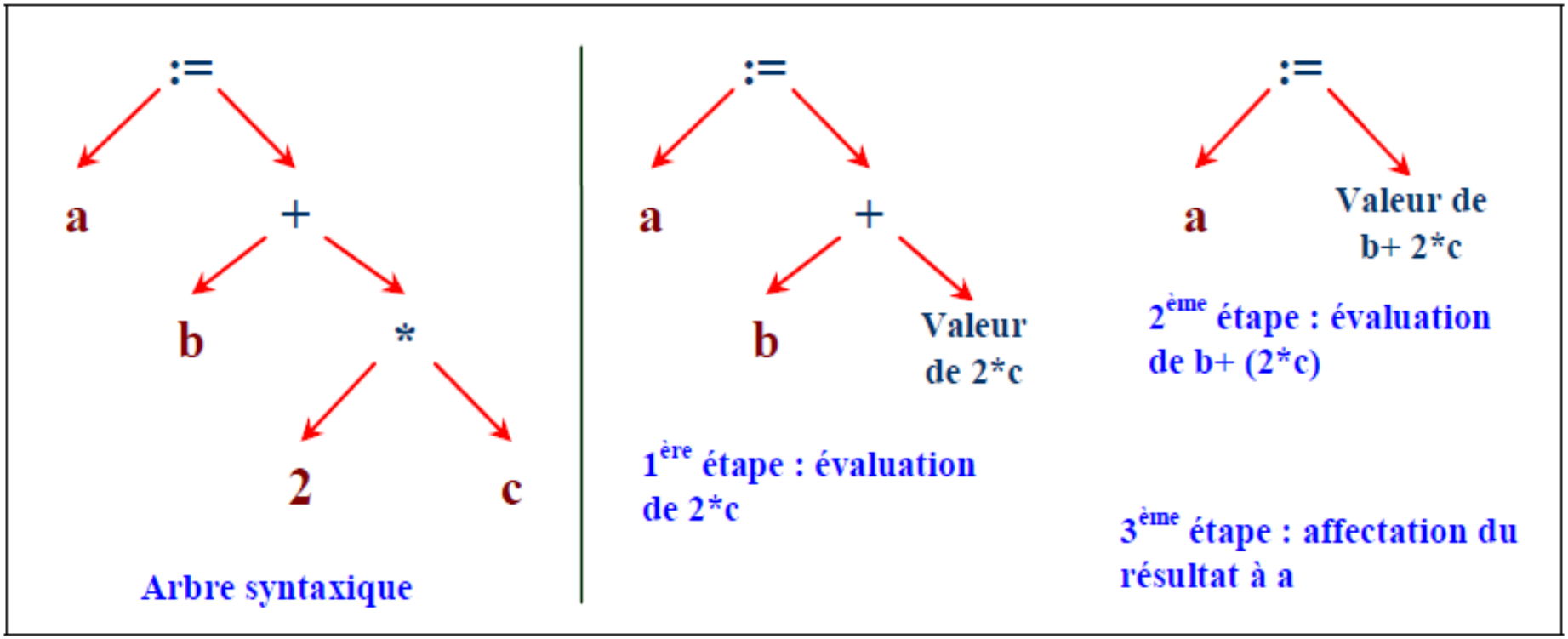
L’analyseur syntaxique

- L’analyseur syntaxique (appelé Parser en anglais) a pour rôle principal la vérification de la syntaxe du code en regroupant les unités lexicales suivant des structures grammaticales qui permettent de construire une représentation syntaxique du code source.

- Notons que durant cette phase, des informations, telles que le type des identificateurs, sont enregistrées dans la table des symboles

Exemple

- La représentation sous forme d'arbre syntaxique de l'expression « a := b + 2 * c ; » est donnée par la figure suivant:



L’analyseur sémantique

- Il a comme rôle principal le contrôle du code source, pour détecter éventuellement l’existence d’erreurs sémantiques, et la collecte des informations destinées à la production du code intermédiaire.

- Un des constituants importants de la phase d’analyse sémantique est le contrôle du type qui consiste à vérifier si les opérandes de chaque opérateur sont conformes aux spécifications du langage utilisé pour le code source.

Exemple

- Dans l’analyse sémantique de « **a := b + 2 * c ; »,** il faut vérifier que, si a est de type entier, alors b et c le sont aussi, sinon il faut signaler une erreur.

- Si on suppose que a, b et c sont de type réel, alors pendant l’évaluation de l’expression, l’analyse sémantique aura comme tâche d’insérer une opération de conversion de type pour transformer la valeur entière 2 en valeur réelle 2.0.

- Cela peut être effectué sur l’arbre syntaxique comme le montre la figure suivant

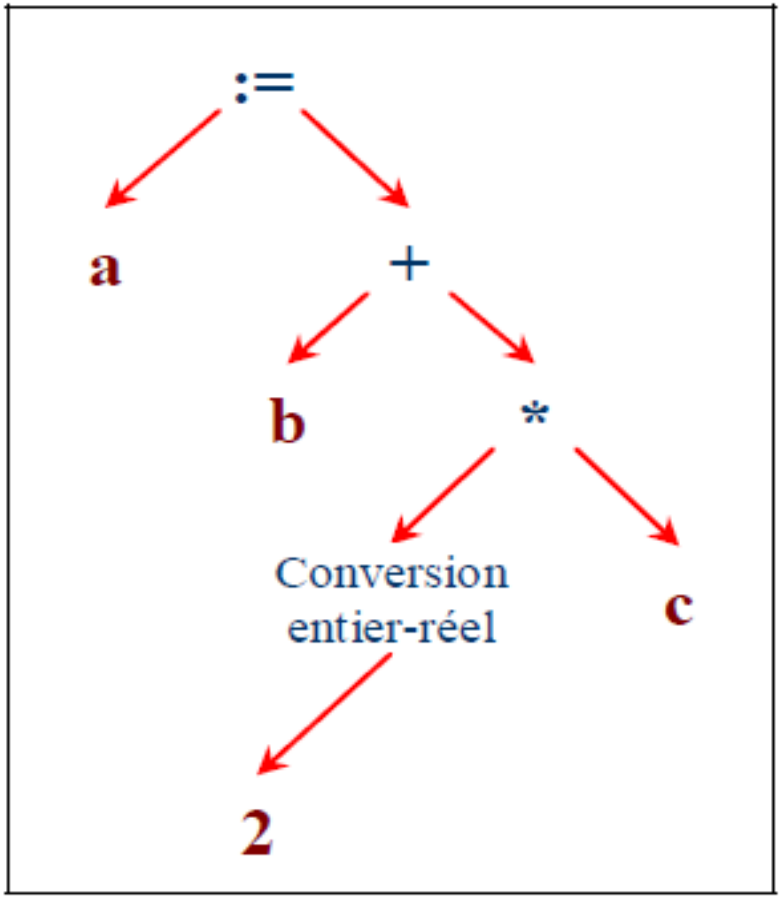
L’analyseur sémantique

- Il a comme rôle principal le contrôle du code source, pour détecter éventuellement l’existence d’erreurs sémantiques, et la collecte des informations destinées à la production du code intermédiaire.

- Un des constituants importants de la phase d’analyse sémantique est le contrôle du type qui consiste à vérifier si les opérandes de chaque opérateur sont conformes aux spécifications du langage utilisé pour le code source.

L’analyseur sémantique

- Enrichissement de l’arbre syntaxique lors de la phase d’analyse sémantique



Le générateur de code intermédiaire

- Certains compilateurs construisent explicitement une représentation intermédiaire du code source sous forme d'un code intermédiaire qui n'est pas directement exécutable par une machine spécifique.

- C'est plutôt un code généré pour une machine abstraite (virtuelle), qui a la double caractéristique d'être, à la fois, facile à produire, à partir de l'arbre syntaxique, et facile à convertir pour une machine réelle donnée.

Le générateur de code intermédiaire

Exemple

- Nous avons supposé que la machine abstraite est une machine à une adresse qui dispose d'un seul accumulateur et dont le jeu d'instruction contient les instructions LoadValue, ConvReal, Mul, Add et Store.

- Elles permettent de réaliser les opérations données dans la deuxième colonne. Nous avons supposé aussi que a occupe la première entrée dans la table des symboles, b occupe la deuxième et c la troisième.

- Ainsi, le code intermédiaire de l'expression « a := b + 2 * c ; », peut être comme suit :

Le générateur de code intermédiaire

Code	Signification opérationnelle des instructions
LoadValue 2	Charger l'accumulateur avec une valeur directe (2)
ConvReal	Convertir le contenu de l'accumulateur en réel
Store temp1	Stocker le résultat dans la variable temporaire temp1
Load temp1	Charger l'accumulateur avec la valeur de temp1
Mul 3	Multiplier le contenu de l'accumulateur par le contenu de l'entrée 3 de la table de symboles (c)
Add 2	Additionner le contenu de l'accumulateur au contenu l'entrée 2 de la table de symboles (b)
Store 1	Ranger le contenu de l'accumulateur dans l'entrée 1 de la table de symboles (a)

L’optimiseur du code intermédiaire

- Lors de la phase d’optimisation, le code intermédiaire est changé pour améliorer les performances du code cible qui en sera généré.

- Il s’agit principalement de réduire le temps d’exécution et l’espace mémoire qui sera occupé par le code cible.

- L’optimisation supprime, par exemple, les identificateurs non utilisés, élimine les instructions inaccessibles, élimine les instructions non nécessaires, fait ressortir hors des boucles les instructions qui ne dépendent pas de l’indice de parcours des boucles, etc.

L’optimiseur du code intermédiaire

Exemple

- On constate dans l’exemple précédent que la valeur convertie 2.0 est stockée dans temp1 puis récupérée et chargée dans l’accumulateur. Puisque aucun usage n’est fait de temp1 dans le reste du code, il est possible d’éliminer les deux instructions en question. Après conversion, le résultat 2.0 reste alors dans l’accumulateur et sera utilisé directement dans la multiplication.

- Noter que, à l’opposé, si la conversion en réel de la valeur 2 est souvent nécessaire, il serait préférable de la stocker dans un espace temporaire. Cependant, ce dernier augmente l’espace réservé aux données dans le code cible.

- Le nouveau code intermédiaire est le suivant :

L’optimiseur du code intermédiaire

Code	Signification opérationnelle des instructions
LoadValue 2	Charger l’accumulateur avec une valeur directe
ConvReal	Convertir le contenu de l’accumulateur en réel
Mul 3	Multiplier le contenu de l’accumulateur par le contenu de l’entrée 3 de la table de symboles
Add 2	Additionner le contenu de l’accumulateur au contenu l’entrée 2 de la table de symboles
Store 1	Ranger le contenu de l’accumulateur dans l’entrée 1 de la table de symboles

Le générateur du code cible

- C’est la phase finale d’un compilateur qui consiste à produire du code cible dans un langage d’assemblage ou un langage machine donné. Le code généré est directement exécuté par la machine en question ou alors il l’est après une phase d’assemblage.

Le générateur du code cible

- Considérons une machine à deux adresses qui dispose de deux registres de calcul R1 et R2. Nous supposons que les variables a, b et c de la table des symboles ont comme adresses de cellules mémoires correspondantes ad1, ad2 et ad3. Le code cible qui sera généré pour cette machine est donné dans le tableau suivant :

Code	Signification opérationnelle des instructions
MOV R1, #2	Charger R1 avec la valeur directe 2
CReal R1	Convertir le contenu de R1 en réel
MOV R2, ad3	Charger le registre R2 avec le contenu de la cellule mémoire d'adresse ad3
MUL R1, R2	Multiplier le contenu de R1 par le contenu de R2, le résultat est dans le premier registre
MOV R2, ad2	Charger R2 avec le contenu de l'adresse ad2
ADD R1, R2	Additionner le contenu de R1 au contenu de R2, le résultat est dans le premier registre
STO R1, ad1	Ranger le contenu de R1 dans la cellule mémoire d'adresse ad1

Le gestionnaire de la table de symbole

- Les phases logiques de compilation échangent des informations par l'intermédiaire de la table des symboles.

- C'est une structure de données (généralement une table) contenant un enregistrement pour chaque identificateur utilisé dans le code source en cours d'analyse.

- L'enregistrement contient, parmi d'autres informations, le nom de l'identificateur, son type, et l'emplacement mémoire qui lui correspondra lors de l'exécution.

- A chaque fois que l'analyseur lexical rencontre un identificateur pour la première fois, le gestionnaire de la table des symboles insère un enregistrement dans la table et l'initialise avec les informations actuellement disponibles (le nom).

- Lors de l'analyse syntaxique, le gestionnaire associera le type à l'identificateur, alors que, lors de l'analyse sémantique, une vérification de types est opérée grâce à cet enregistrement.

Le gestionnaire des erreurs

- Son rôle est de signaler les erreurs qui peuvent exister dans le code source et qui sont détectées lors des différentes phases logiques de la compilation. Il doit produire, pour chaque erreur, un diagnostic clair et sans ambiguïté qui permettra la localisation et la correction de l'erreur par l'auteur du code source.