Chapitre 1 : compilation (Introduction)

Introduction

Les programmeurs écrivent généralement un programme informatique utilisant un langage haut niveau (langage de programmation). Mais un ordinateur ne comprend pas le langage haut niveau. Il ne comprend que les programmes écrits en binaire 0 et 1 appelé le code machine. Un programme écrit en langage évolué s'appelle un code source. Nous devons convertir le code source en code machine, ce que font les compilateurs et les interpréteurs. Par conséquent, un compilateur ou un interpréteur est un programme qui convertit un programme écrit en langage évolué en un code machine compris par l'ordinateur.

Objectif du cours :

1. Compréhension du cheminement d'un programme (texte) source vers un programme (code) en langage machine.

2. Étude des étapes du processus de compilation d'un langage évolué.

3. les principes de base inhérents à la réalisation de compilateurs : analyse lexicale, analyse syntaxique, analyse sémantique, génération de code.

4. les outils fondamentaux utilisés pour effectuer ces analyses : principe de base de la théorie des langages (expressions régulières, automates, grammaires)

5. Familiarisation, en TP, avec des outils de génération d'analyseurs lexicaux et syntaxiques (LEX et YACC).

I. Définition d'un compilateur

Tout programme écrit dans un langage de haut niveau ne peut être exécute par un ordinateur que s'il est traduit en instructions exécutables par l'ordinateur (langage machine : instructions élémentaires directement exécutables par le processeur.

Donc, Un compilateur est un logiciel particulier qui traduit un programme écrit dans un langage de haut niveau (par le programmeur) en instructions exécutables (par un ordinateur).

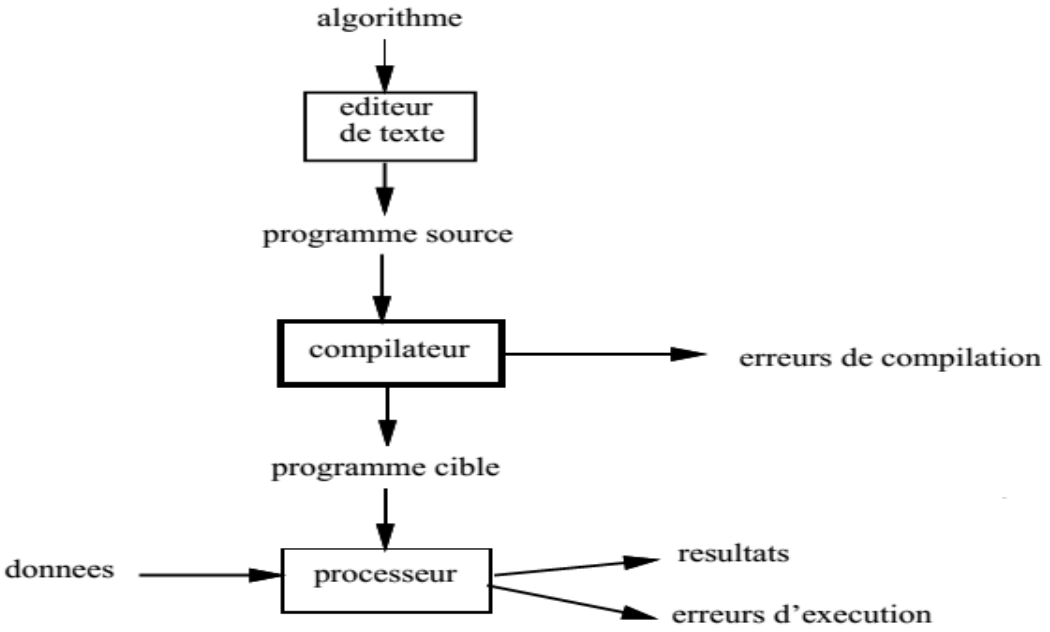

Figure 1 : chaîne de développement d'un programme

II. Structure d'un compilateur

La compilation se décompose en deux phases :

1- Une phase d'analyse : qui va reconnaitre les variables, les instructions, les opérateurs et vérifier la structure syntaxique du programme ainsi que certaines propriétés sémantiques. Cette phase possède trois parties (analyse lexicale, analyse syntaxique, analyse sémantique)

2- Une phase de production : qui devra produire le code cible. Il possède deux parties : génération de code et optimisation de code.

La phase d'analyse

1) Analyse lexicale : 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).

Par exemple à partir du morceau de C suivant :

If (i<a+b) x=x*2;

L'analyseur lexical déterminera la suite de token :

Les tokens	Signification
If	Mot clé
(	séparateur
i	identificateur
<	opérateur relationnel
a	identificateur

+	opérateur arithmétique
b	identificateur
)	séparateur
x	identificateur
=	affectation
x	identificateur
*	opérateur arithmétique
2	constant
;	séparateur

2) Analyse syntaxique

Lors de l'analyse syntaxique, on vérifie que l'ordre des tokens correspond à l'ordre définit pour le langage. On dit que l'on vérifie la syntaxe du langage à partir de la définition de sa grammaire. L'analyseur syntaxique sait comment doivent être construites les expressions, les instructions, les déclarations de variables, les appels de fonctions…

En autre terme, L'analyseur syntaxique 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. Cette dernière a souvent une structure en arbre.

Exemple : En C, un exemple d'une instruction doit se présenter sous la forme:

if (expression) instruction

Si l'analyseur syntaxique reçoit la suite d'unités lexicales :

Mot clé identificateur opérateur relationnel constant séparateur identificateur affectation identificateur opérateur arithmétique identificateur séparateur

→il doit signaler que ce n'est pas correct car il n'y a pas de **(** juste après le if.

Mais si l'analyseur syntaxique reçoit la suite d'unités lexicales :

mot clé séparateur identificateur opérateur relationnel constant séparateur identificateur affectation identificateur opérateur arithmétique constant séparateur

→L'analyseur syntaxique signale que cette suite des unités lexicales est syntaxiquement correcte.

3) Analyse sémantique ou vérification de type

Dans cette phase on vérifie que les variables ont un type correct. Par exemple, il faut vérifier que la variable 'i' possède bien le type 'entier', et que la variable 'a' est bien un nombre. Et aussi, on ne peut pas par exemple : additionner un réel avec une chaine de caractères, ou affecter une variable à un nombre ….

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.

La phase de production

1) Génération de code : Le générateur de code est la phase finale de compilateur, il a pour donner une représentation intermédiaire du programme source et produit comme résultat un programme cible équivalent, qui doit être correct et de bonne qualité, de plus doit s'exécuter le plus rapidement possible.

2) Optimisation de code : Il s'agit d'améliorer le code produit afin que le programme résultant soit plus rapide.

Il y a des optimisations qui ne dépendent pas de la machine cible : élimination de calculs inutiles, propagation des constantes, extraction des boucles des invariants de boucle.

Et il y a des optimisations qui dépendent de la machine cible : remplacer des instructions générales par des instructions plus efficaces et plus adaptées, utilisation optimale des registres

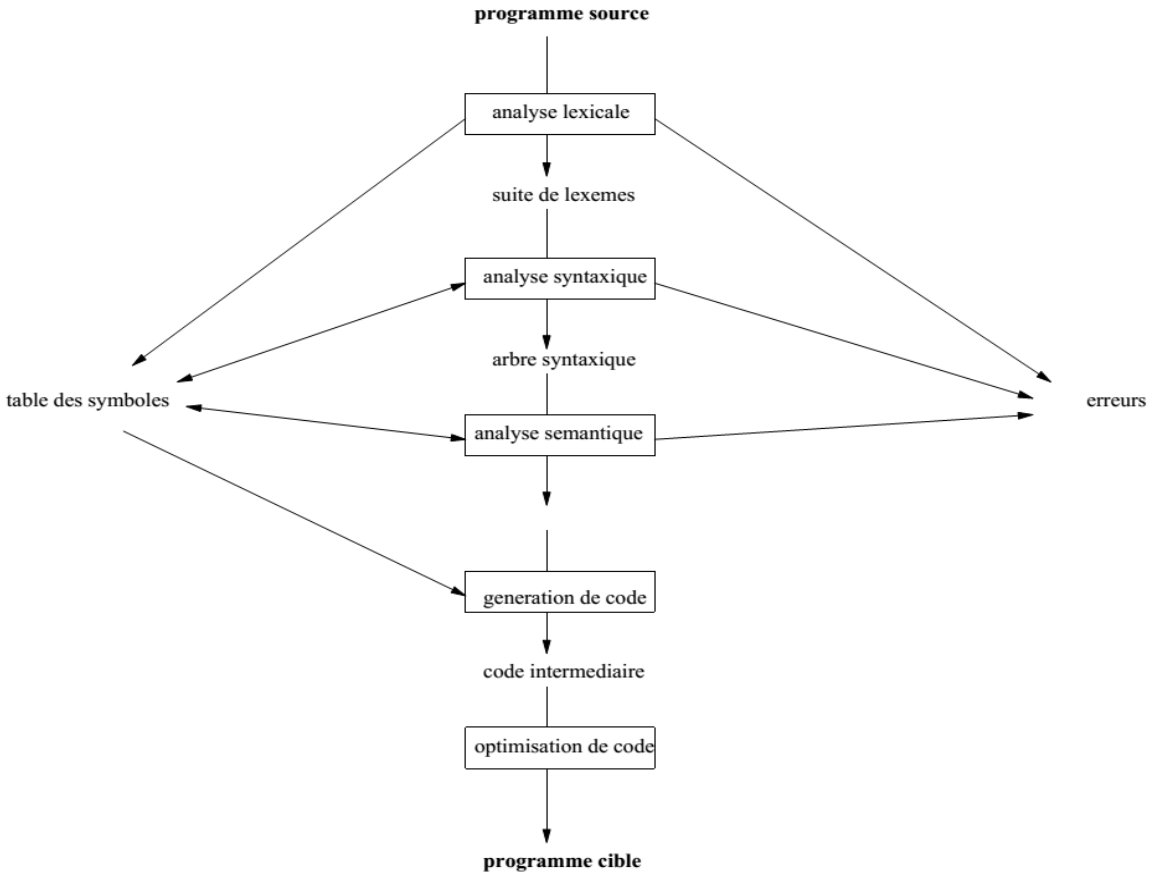

Figure 2 : Structure d'un compilateur

III. Interpréteur versus Compilateur

→**Un interpréteur** est un logiciel qui interprète le programme source, pendant son exécution. C'est-à-dire qu'il analyse les instructions du programme source, les unes après les autres, et exécute chacune d'elles immédiatement. Dans ce cas, il n'y a pas de création d'un programme objet équivalent.

→**Un compilateur** est un programme informatique qui traduit l'ensemble du code source d'un projet logiciel en code machine avant son exécution. C'est uniquement après cette traduction que le projet sera exécuté par le processeur qui dispose de toutes les instructions sous forme de code machine.

interpréteur	compilateur
Convertit le programme en prenant ligne après l'autre	convertir l'intégralité du code
Aucun code d'objet n'est généré	Génère du code d'objet
Continue de traduire le programme jusqu'à ce que la première erreur soit rencontrée	Il génère le message d'erreur uniquement après avoir analysé l'ensemble du programme (de façon groupée à la fin de la compilation)
La traduction du code source est pendant le fonctionnement du logiciel	La traduction du code source est avant l'exécution du logiciel
Exemple d'interpréteur : PHP, Python….	Exemple de compilateur : C, C++