Module : Compilation

Chapitre 1 : Introduction générale

1.1 introduction

Le but de ce cours est de présenter les principes de base inhérents à la réalisation de compilateurs. Les idées et techniques développées dans ce domaine sont si générales et fondamentales qu'un informaticien les utilisera très souvent au cours de sa carrière : traitement de données, moteurs de recherche, etc. ..

Comprendre comment est écrit un compilateur permet de mieux comprendre les "contraintes" imposées par les différents langages lorsque l'on écrit un programme dans un langage de haut niveau.

Définition : Un compilateur est un programme qui lit un programme écrit dans un premier langage (programme source) et le traduit en un programme équivalent écrit dans un autre langage (cible). Pendant la traduction, le compilateur signale les éventuelles erreurs à l'utilisateur.

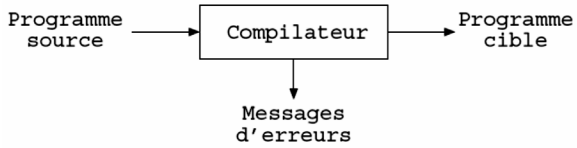

FIGURE 1.1 – Un compilateur.

La compilation fait appel à :

- la théorie des langages ;
- l'architecture des machines ;
- l'algorithmique et le génie logiciel.

1.2 Environnement du compilateur

Au début des années 50, la réalisation d'un compilateur est difficile. Aujourd'hui : utilisation de techniques systématiques. La figure 1.2 montre l'environnement du compilateur .

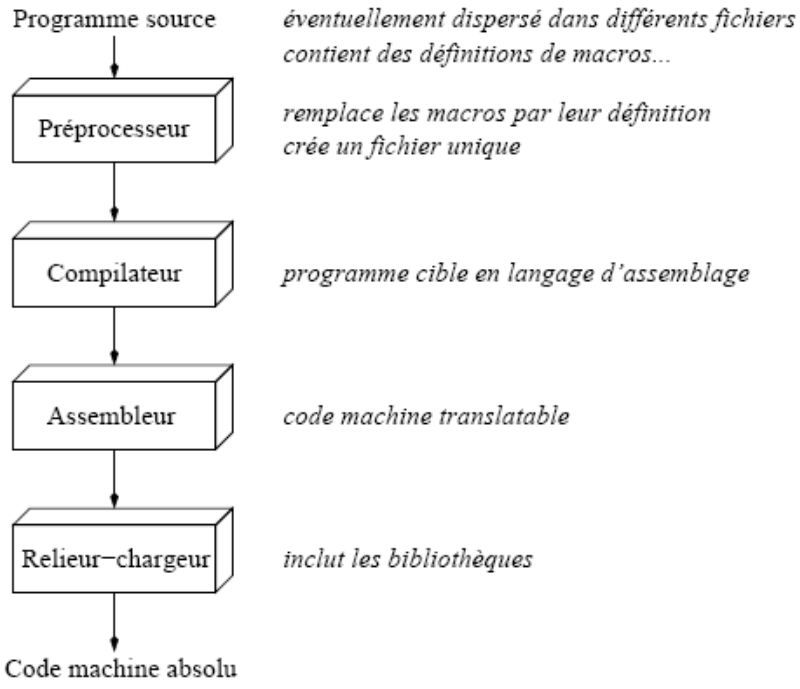

Figure 1.2 : Environnement du compilateur

1.3 Phases d'un compilateur

La compilation est décomposée de deux grandes phases :

- la phase d'analyse, son rôle est reconnaître les variables, les instructions, les opérateurs et élaborer la structure syntaxique du programme, et ainsi que certaines propriétés sémantiques.
- la phase de synthèse, qui construit le programme cible désiré en utilisant une représentation intermédiaire produite par la phase de l'analyse.

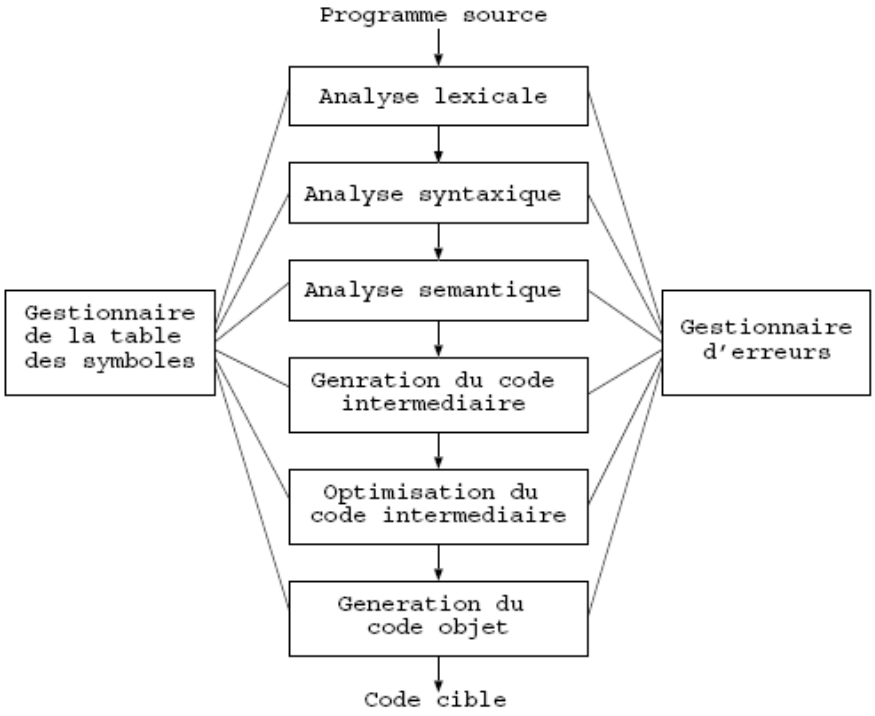

FIGURE 1.2 – Structure logique d'un compilateur.

La figure 1.3 montre la structure logique d'un compilateur.

1.3.1 La phase analyse

1. **Analyse lexicale (linéaire) :** sa tâche principale est lire les caractères d'entrée et de produire comme résultat une suite d'unités lexicales que la deuxième phase (analyse syntaxique) aura à traiter. Elle s'occupe ainsi de l'élimination des caractères superflus (blancs, tabulations, commentaires…etc)
   Outils théoriques utilisés : **expressions régulières** et **automates à états finis**
2. **Analyse syntaxique (hiérarchique ou grammaticale) :** consiste à regrouper les unités lexicales du programme source en structures grammaticales (arbre syntaxique) qui seront employées par le compilateur pour synthétiser son résultat.
   Outils théoriques utilisés : **grammaires type 2 et automates à pile**

3. **Analyse sémantique (analyse contextuelle) :** dans cette phase, on opère certains contrôles (contrôles de type, par exemple) afin de vérifier que l'assemblage des constituants du programme a un sens.
   Outil théorique utilisé : **schéma de traduction dirigée par la syntaxe**.
   A titre d'exemple on doit vérifier si :

   - 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 à insérer ?

- les arguments des appels de fonctions ont-ils le nombre et le type requis ?

1.3.2 La phase de synthèse

1. Production du code intermédiaire : après l'analyse sémantique, certains compilateurs construisent une représentation intermédiaire du programme source. Cette représentation doit vérifier deux propriétés : facile à produire et facile à traduire en langage cible.

2. Optimisation du code : tente d'améliorer le code intermédiaire de façon que le code résultat s'exécuter plus rapidement et/ou consomme moins d'espace mémoire. Cette phase peut consommer une part significative du temps de la compilation.

3. Production du code : consiste à produire un code en langage d'assemblage. Lors de cette étape, on sélectionne des emplacements mémoire pour chacune des variables utilisées. Puis les instructions (du code intermédiaire) sont traduites en suites d'instructions (en langage d'assemblage) effectuant la même tâche.

Remarque :

La table des symboles est une structure de données contenant un enregistrement pour chaque identificateur, muni de champs pour les attributs de l'identificateur (adresse, type, portée, type de retour (pour les fonctions), etc.).

Après avoir détecté une erreur par une phase de compilation, cette dernière doit la traiter de telle façon que la compilation puisse continuer et que d'autres erreurs puissent être détectées (un compilateur qui s'arrête à la première erreur n'est pas très utilisable).

Exemple :

La figure 1.4 présente un exemple de compilation d'une instruction d'affectation.

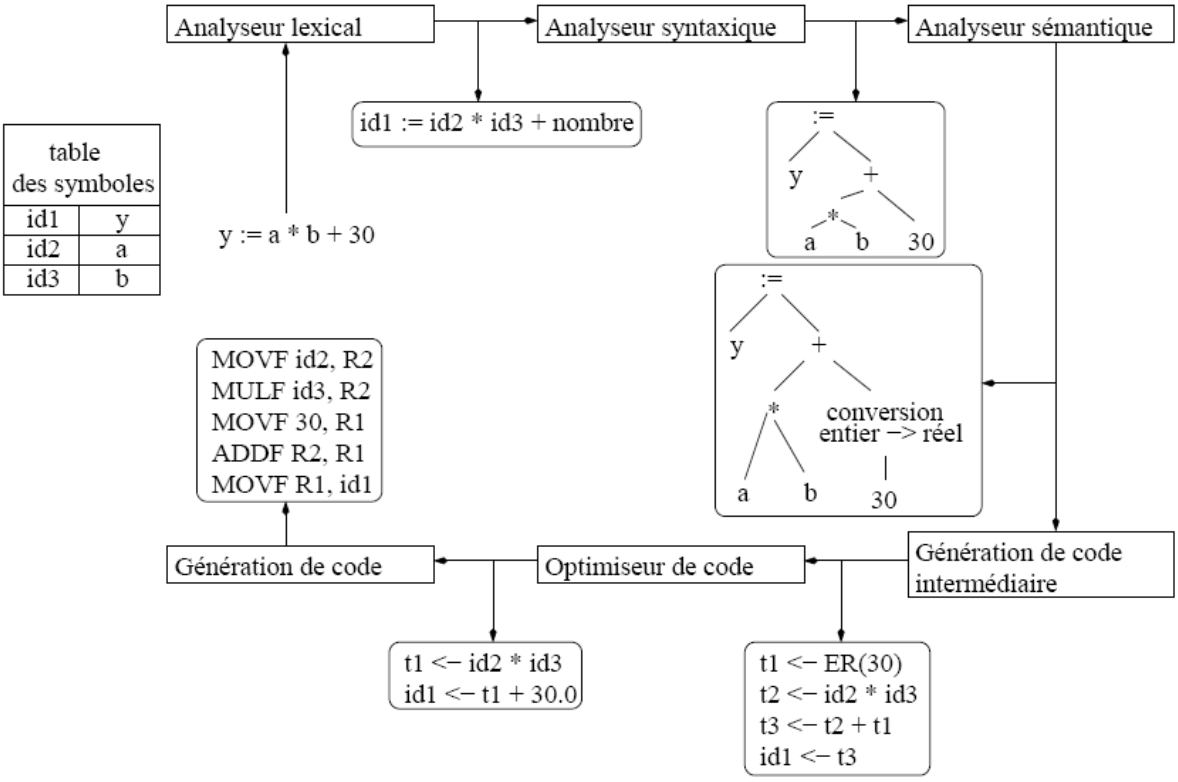

Figure 1.4 : compilation d'une simple instruction

1.4 Compilateur et interpréteur

- Au lieu de produire un programme cible, un **interprète** exécute lui-même au fur et à mesure les opérations spécifiées par le programme source.
- Un interpréteur analyse une instruction après l'autre puis l'exécute immédiatement.
- A l'inverse d'un compilateur, un interpréteur travaille simultanément sur le programme et sur les données.
- L'interpréteur doit être présent sur le système à chaque fois que le programme est exécuté, ce qui n'est pas le cas avec un compilateur.
- Généralement les interpréteurs sont assez petits, mais le programme est plus lent qu'avec un langage compilé.
- Les langages interprétés sont souvent plus simples à utiliser et tolèrent plus d'erreurs de codage que les langages compilés.