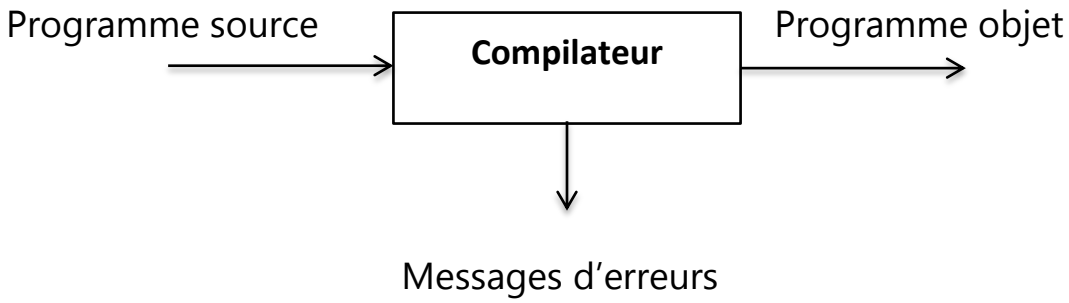
I. Introduction à la compilation :

1. Notion de traducteur :

Un traducteur est un programme qui transforme un programme source en un programme objet sémantiquement équivalent. "Sémantiquement équivalent", veut dire que le programme obtenu doit assurer les mêmes tâches et fournir les mêmes résultats que ceux du programme traduit.

2. Le compilateur :

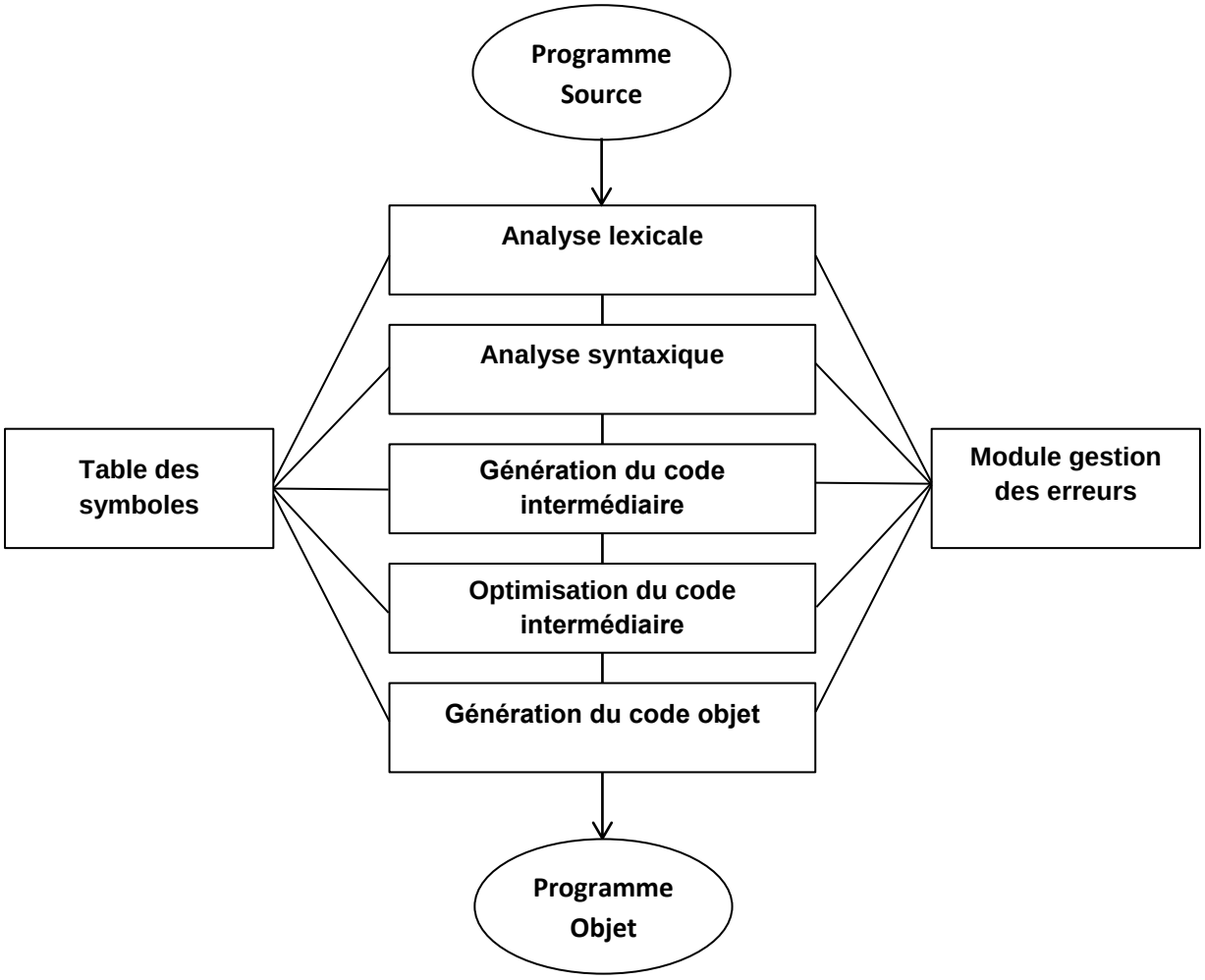
Le compilateur est un traducteur qui transforme un programme source écrit en un langage évolué (C, C++, Pascal, Java,…) en un programme objet équivalent écrit en code machine. Le compilateur doit être capable de détecter les erreurs qui peuvent être commises dans le programme source et de les signaler pour qu'elles soient corrigées avant de générer le code objet (code machine).



3. Structure d'un compilateur :

Généralement, un compilateur est constitué d'une suite de phases successives qui, à tour de rôle, transforment et traitent le programme considéré jusqu'à l'obtention de la forme finale qui est le code objet. Pour mener à bien ces opérations, ces phases utilisent deux principaux modules qui sont celui de la gestion de la table des symboles et celui de la gestion des erreurs.

Le schéma suivant, résume cette description :



3.1. Différentes phases d'un compilateur :

3.1.1. L'analyse lexicale :

Elle prend en entrée le texte du programme source, le lit caractère par caractère pour former les différentes entités lexicales ou lexèmes (toute suite de caractères juxtaposés situés entre deux séparateurs) et fournit ainsi, la liste de lexèmes (jetons ou tokens), accompagnés chacun, par son unité lexicale à laquelle il appartient et d'autres informations nécessaires pour la suite de l'analyse.

En général, les différentes entités lexicales sont classifiées comme suit :

- Les mots clés : les mots qui sont propres au langage.

- Les identificateurs : les mots qui sont proposés par l'utilisateur pour nommer ses variables, ses constantes, ses procédures,…

- Les constantes

- Les opérateurs : +, -, /, and, or,…

- Les séparateurs : virgule (,), Point-virgule(;), parenthèses,…

3.1.2. L'analyse syntaxique :

Son principal rôle est de vérifier si le programme source est conforme à la syntaxe du langage utilisé pour l'écrire et d'envoyer des messages d'erreur dans le cas échéant. Elle prend la liste des lexèmes produite par l'analyse lexicale et regroupe ces derniers en une structure d'arbre (appelée l'arbre syntaxique) qui reflète la structure grammaticale du programme source. Durant cette phase, la table des symboles est toujours utilisée pour y répertorier les identificateurs rencontrés ou pour la consulter afin de vérifier la présence et le rôle d'un identificateur à analyser.

3.1.3. La production du code intermédiaire :

L'arbre syntaxique obtenu durant la phase précédente est transformé en une représentation intermédiaire simple à l'aide d'un code de bas niveau dont la structure est indépendante de la machine. On peut distinguer plusieurs formes de codes intermédiaires, dont les plus utilisées sont :

- Le code post-fixé : l'avantage de ce code, est que l'ordre des entités qui y figurent est le même que l'ordre de leur exécution par la machine.

- Le code à trois adresses : ce code ressemble au code assembleur où les instructions générées contiennent un seul opérateur mais, à la différence de l'assembleur, ce code ne spécifie pas les registres de la machine.

3.1.4. L’optimisation du code intermédiaire :

Cette phase est optionnelle ; de nombreux compilateurs en font abstraction. Le code intermédiaire produit est amélioré de façon à ce que le code machine résultant soit exécuté le plus rapidement possible surtout dans le cas où l’espace mémoire où le temps machine sont critiques. La minimisation peut considérer la factorisation de variables, de code, "minimisation" du nombre de registres nécessaires, etc…

3.1.5. La production du code objet :

Après optimisation du code intermédiaire, cette étape permet de le convertir en code objet qui peut être soit du code machine ou de l’assembleur. Le problème principal ici, est la nécessité de bien connaître la structure du code objet (jeu d’instructions), ces différents modes d’adressage, ainsi que la structure de la machine sur laquelle va être exécuté le programme obtenu. Une tâche cruciale de cette étape est l’allocation des registres : les noms des variables symboliques utilisés dans le code intermédiaire seront traduits en codes dont chacun correspond à un registre dans le code machine cible.

3.2. Les modules complémentaires :

3.2.1. Le gestionnaire de la table des symboles :

Une table de symbole est une table qui stocke les informations nécessaires sur les lexèmes. Le gestionnaire de la table des symboles est un programme qui gère cette table (construction de la table, insertions et recherches).

3.2.2. Le gestionnaire des erreurs:

Il s’agit du module responsable de la détection et de la signalisation des erreurs rencontrées dans le programme source.

3.2.3. Le pre-processeur :

Certains langages (comme C) ont besoin d’un module qui effectue un pré-traitement du texte source avant de déclencher le processus de compilation.

Le plus souvent, la tâche de ce module consiste à faire des recherches et/ou des remplacements de texte ou d'inclusion de certains fichiers. .

3.2.4. L'assemblage et l'édition de liens :

Le code en langage d'assemblage est traduit en représentation binaire et les adresses des variables et des fonctions sont fixées. Ces deux étapes sont typiquement effectuées par des programmes fournis par la machine ou par le constructeur du système d'exploitation, et ne font pas partie du compilateur lui-même.

3.2.5. Le chargeur :

C'est un programme qui charge le code machine absolu dans la mémoire principale (la RAM) de l'ordinateur.