République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider Biskra

Faculté des Sciences Exactes et des Sciences de la Nature et de la Vie
Département d'informatique

Polycopié : Compilation
Pour 3ème Année Licence
Spécialité : Systèmes informatique

Par

MEADI Mohamed Nadjib, Maitre de conférences B

Année Universitaire 2022/2023

Table Des Matières

Table des matières

CHAPITRE 01 : INTRODUCTION 1

1 INTRODUCTION 2
2 POURQUOI CE COURS ? ..... **ERREUR ! SIGNET NON DEFINI.**
3 COMPILATEUR ET INTERPRETE ..... **ERREUR ! SIGNET NON DEFINI.**
4 STRUCTURE D'UN COMPILATEUR 2
4.1 Une phase d'analyses 3
4.2 Une phase de synthèse 4
4.3 Phase parallèle 4

CHAPITRE 02 : ANALYSE LEXICALE 6

1 INTRODUCTION 7
2 DEFINITIONS 7
3 SPECIFICATION DES UNITES LEXICALES 8
3.1 Expressions régulières 8
*3.2 Présentation des expressions régulières .....**Erreur ! Signet non défini.***
3.3 Définitions régulières 9
*3.4 Notations abrégées .....**Erreur ! Signet non défini.***
3.5 Automates 9
3.6 Construction d'un AFN à partir d'une E.R. 11
3.7 Automates finis déterministes (AFD) 11
4 MISE EN ŒUVRE D'UN ANALYSEUR LEXICAL 11
5 OUTIL LEX (GENERATEUR DES ANALYSEURS LEXICAUX) 13
5.1 Les expressions régulières Lex 14
5.2 Structure d'un fichier Lex 14
5.2.1 Partie des déclarations 15
5.2.2 Partie des productions 15
5.2.3 Section Procédures auxiliaires 16
5.3 Variables et fonctions prédéfinies 16
6 ERREURS LEXICALES 17
7 EXERCICES 18

CHAPITRE 03 : ANALYSE SYNTAXIQUE 20

1 INTRODUCTION 21
2 GRAMMAIRES ET ARBRES DE DERIVATION 21
2.1 Grammaires hors contexte 21
2.2 Arbre de dérivation 21
3 MISE EN ŒUVRE D'UN ANALYSEUR SYNTAXIQUE 23
3.1 Analyse descendante 23
3.1.1 Analyse par descente récursive 23
3.1.2 Récursivité à gauche 26
3.1.3 Factorisation à gauche 27
3.1.4 Grammaire propre 28
3.1.5 Table d'analyse LL(1) 28
3.1.6 Analyseur syntaxique 30
3.1.7 Grammaire LL(1) 31
4 EXERCICES 33
5 ANALYSE ASCENDANTE 34
5.1 Table d'analyse ascendante (SLR) 35
5.1.1 Construction de la table d'analyse : 35
5.2 Analyseur syntaxique SLR : 37
5.3 Analyseur syntaxique LR(1) 38
5.4 Analyseur syntaxique LALR(1) 39
6 L'OUTIL YACC 41
6.1 La structure d'une spécification Yacc 41
6.1.1 Partie Déclaration 42
6.1.2 Grammaire et actions 42
6.1.3 Le programme en C 42
6.2 Communication avec l'analyseur lexical : yylval 42

6.3 Variables, fonctions et actions prédéfinies 43
6.4 Etude de cas: 44
7 ERREURS SYNTAXIQUES 45
7.1 Récupération en mode panique 46
7.2 Récupération au niveau du syntagme 46
7.4 Productions d'erreur 47
7.5 Correction globale 47

CHAPITRE 04 : ANALYSE SEMANTIQUE 49

1 INTRODUCTION 50
2 DEFINITION DIRIGEE PAR LA SYNTAXE 50
2.1 Les attributs 51
2.2 Graphe de dépendances 52
2.3 Construction des arbres abstraits 53
3 ORDRE D'EVALUATION 53
3.1 Après l'analyse syntaxique 53
3.2 Pendant l'analyse syntaxique 53
4 SCHEMA DE TRADUCTION 55
5 PORTEE DES IDENTIFICATEURS 56
6 CONTROLE DE TYPE 57
7 SURCHARGE D'OPERATEURS ET DE FONCTIONS 59
8 FONCTIONS POLYMORPHES 59

CHAPITRE 05 : GENERATION DU CODE 60

1 INTRODUCTION 61
2 ENVIRONNEMENT D'EXECUTION 61
2.1 Les objets statiques 61
2.2 Les objets automatiques 61
2.3 Les objets dynamiques 62
3 CODE INTERMEDIAIRE 63
4 CARACTERISTIQUES COMMUNES AUX MACHINES CIBLES 63
5 CODE A 3 ADRESSES SIMPLIFIE 63
5.1 Production de code à 3 adresses 65
6 GRAPHE DE FLOT DE CONTROLE 68

REFERENCES 70

Liste des figures

FIGURE 1.1: LES DIFFÉRENTES PHASES COMPOSANTS UN COMPILATEUR 2
FIGURE 1.2: EXEMPLES DE DIFFÉRENTES ÉTAPES COMPOSANT UN COMPILATEUR 5
FIGURE 2.1 : LES ENTÉES ET LES SORTIES D'UN ANALYSEUR LEXICALE 7
FIGURE 2.2 : LA COOPÉRATION ENTRE L'ANALYSEUR LEXICALE ET L'ANALYSEUR SYNTAXIQUE 7
FIGURE 2.3: REPRÉSENTATION GRAPHIQUE D'UN AUTOMATE 10
FIGURE 2.4 : EXEMPLE D'UN AUTOMATE NON DITERMINSTE ..... **ERREUR ! SIGNET NON DEFINI.**
FIGURE 2.5 : AUTOMATE FINI DÉTERMINISTE OBTENU L'APPLICATION DE L'ALGORITHME DE DÉTEMINISATION ..... **ERREUR ! SIGNET NON DEFINI.**
FIGURE 2.6: AUTOMATE MINIMUM ..... **ERREUR ! SIGNET NON DEFINI.**
FIGURE 2.7: EXEMPLE D'UN AUTOMATE ..... **ERREUR ! SIGNET NON DEFINI.**
FIGURE 2.8 : EXEMPLE D'UN AUTOMATE SPÉCIFIANT QUELQUES OPÉRATEURS ARITHMÉTIQUES ET LOGIQUES 12
FIGURE 2.9 : LES ÉTAPES À SUIVRE POUR OBTENIR UN ANALYSEUR LEXICAL À PARTIR D'UN FICHIER LES 13
FIGURE 3.1 : DEUX ARBRES DIFFÉRENTS POUR LA MÊME CHAINE. 23
FIGURE 3.2: ARBRE SYNTAXIQUE RÉSULTANTS DE L'ANALYSE DE L'EXPRESSION 3+4*5$ 31
FIGURE 3.3 : ARBRE RÉSULTANT DE L'ANALYSE ASCENDANTE 35
FIGURE 3.4 : L'ARBRE RÉSULTANT DE L'ANALYSE SLR(1) 37
FIGURE 3.5: DÉROULEMENT DE L'ANALYSE SYNTAXIQUE RN UTILISANT YACC 41
FIGURE 4.1 : EXEMPLE D'UNE PILE D'EXÉCUTION 56
FIGURE 5.1 : STRUCTURE DE L'ENVIRONNEMENT DE L'EXÉCUTION 62

Chapitre 01 : Introduction

1 Introduction

Un compilateur est un programme de traduction automatique d'un programme écrit dans un langage source en un programme écrit dans un langage cible .Fondamentalement, c'est une traduction : un texte écrit en langage de programmation évolue, tel que : Pascal, 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.

Une autre phase importante qui intervient après la compilation pour obtenir un exécutable est la phase d'éditions de liens. Un éditeur de liens résout entre autres les références à des appels de routines dont le code est conservé dans des librairies. En général, un compilateur comprend une partie éditeur de liens.
En outre, sur les systèmes modernes, l'édition des liens est faite à l'exécution du programme ! (le programme est plus petit et les mises à jour plus faciles).

2 Structure d'un compilateur

La compilation se décompose en deux phases :

- Une phase d'analyses, qui va reconnaître les variables, les instructions, les opérateurs et élaborer la structure syntaxique du programme ainsi que certaines propriétés sémantiques
- Une phase de synthèse et de production qui devra produire le code cible:

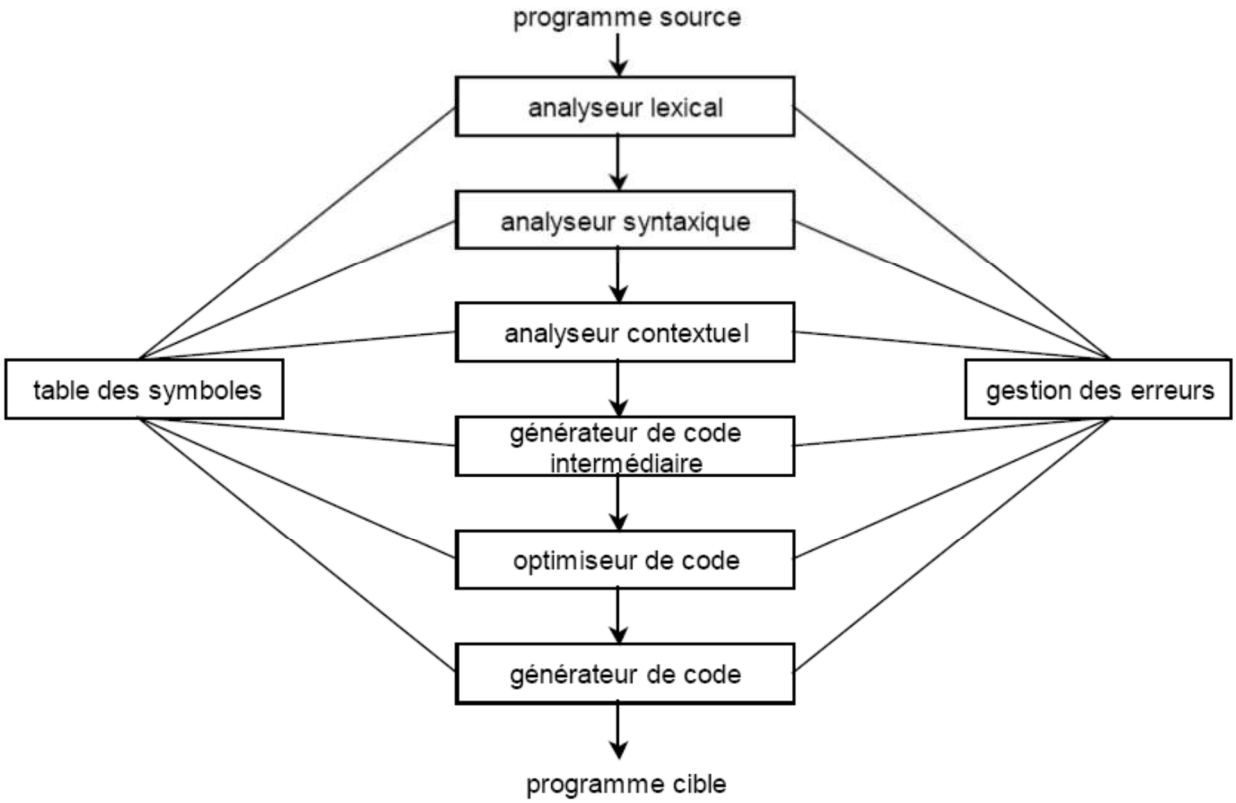

Figure 1.1: Les différentes phases composants un compilateur

2.1 Une phase d'analyses

1. **Analyse lexicale** (appelée aussi Analyse linéaire)

L'objectif de cette étape est de reconnaître les "types" des "mots" lus par le compilateur. La lecture du programme source se fait de gauche à droite et les caractères lus seront regroupés en unités lexicales.

L'analyse lexicale se charge de

- L'élimnation des caractères superflus (commentaires, espaces, ...)
- identifier les parties du texte qui ne font pas partie à proprement parler du programme mais sont des directives pour le compilateur
- identifier les symboles qui représentent des identificateurs, des constantes réelles, entière, chaînes de caractères, des opérateurs (affectation, addition, ...), des séparateurs (parenthèses, points virgules, ...), les mots clefs du langage, ... C'est cela que l'on appelle des unités lexicales.

Outils théoriques utilisés : expressions régulières et automates à états finis

Exemple

Reprenons l'exemple de l'affectation : temps := sec + minute * 60.

La phase d'analyse lexicale va lire la séquence de caractères correspondant à l'affectation. Elle va éliminer les marques de ponctuation et les séparateurs de mots. Ainsi, elle va nous donner dans l'ordre la liste d'unités lexicales suivante :

1. L'identificateur temps ;
2. Le symbole d'affectation := ;
3. L'identificateur sec ;
4. Le signe plus (+) ;
5. L'identificateur minute ;
6. Le signe de multiplication (*) ;
7. Le nombre 60.

Le texte source comporte donc sept unités lexicales, qu'on appelle aussi lexèmes.

2. **Analyse syntaxique** (appelée aussi Analyse hiérarchique ou Analyse grammaticale)

Il s'agit de regrouper les unités lexicales en structures grammaticales, de découvrir la structure du programme. L'analyseur syntaxique sait comment doivent être construites les expressions, les instructions, les déclarations de variables, les appels de fonctions, ...

Outils théoriques utilisés : grammaires et automates à pile.

3. **Analyse sémantique** (appelée aussi 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

- 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 ?
- etc.

2.2 Une phase de synthèse

4. **Génération du code intermédiaire** Il s'agit de produire les instructions en langage cible.

En règle générale, le programmeur dispose d'un calculateur concret (cartes équipées de processeurs, puces de mémoire, ...). Le langage cible est dans ce cas défini par le type de processeur utilisé.

Mais si l'on écrit un compilateur pour un processeur donné, il n'est alors pas évident de porter ce compilateur (ce programme) sur une autre machine cible.

Pour cela, on introduit des machines dites abstraites qui font l'abstraction des architectures réelles existantes. Ainsi, on s'attache plus aux principes de traduction, aux concepts des langages, qu'à l'architecture des machines.
En général, on produira dans un premier temps des instructions pour une machine abstraite (virtuelle). Puis ensuite on fera la traduction de ces instructions en des instructions exécutables par la machine réelle sur laquelle on veut que le compilateur s'exécute. Ainsi, le portage du compilateur sera facilité, car la traduction en code cible virtuel sera faite une fois pour toutes, indépendamment de la machine cible réelle. Il ne reste plus ensuite qu'à étudier les problèmes spécifiques à la machine cible, et non plus les problèmes de reconnaissance du programme (cf Java).

5. **Optimisation du code** Cette phase tente d'améliorer le code produit de telle sorte que le programme résultant soit plus rapide. Par Exemple
    - détecter l'inutilité de recalculer des expressions dont la valeur est déjà connue,
    - transporter à l'extérieur des boucles des expressions et sous-expressions dont les opérandes ont la même valeur à toutes les itérations
    - détecter, et supprimer, les expressions inutiles
    - etc.
6. **Génération du code machine.** Cette phase nécessite la connaissance de la machine cible (réelle, virtuelle ou abstraite), et notamment de ses possibilités en matière de registres, piles, etc.

2.3 Phase parallèle

7. **Gestion de la table des symboles** La table des symboles est la structure de données utilisée pour stocker les informations qui concernent les identificateurs du programme source (par exemple leurs types, leurs emplacements dans la mémoire, leurs portées, visibilités, nombre et type et mode de passage des paramètres d'une fonction, ..etc.
   Le remplissage de cette table a lieu lors des phases d'analyse. Les informations contenues dans la table des symboles sont nécessaires lors des analyses syntaxique et sémantique, ainsi que lors de la génération de code.

8. **Gestion des erreurs** Chaque phase peut rencontrer des erreurs. Cependant, après avoir détecté une erreur, une phase doit la traiter de telle façon que la compilation puisse continuer et que d'autres erreurs dans le programme source puissent être détectées. Un compilateur qui s'arrête à la première erreur n'est pas non plus très performant. Bien sûr, il y a des limites à ne pas dépasser et certaines erreurs peuvent entrainer l'arrêt de l'exécution du compilateur.

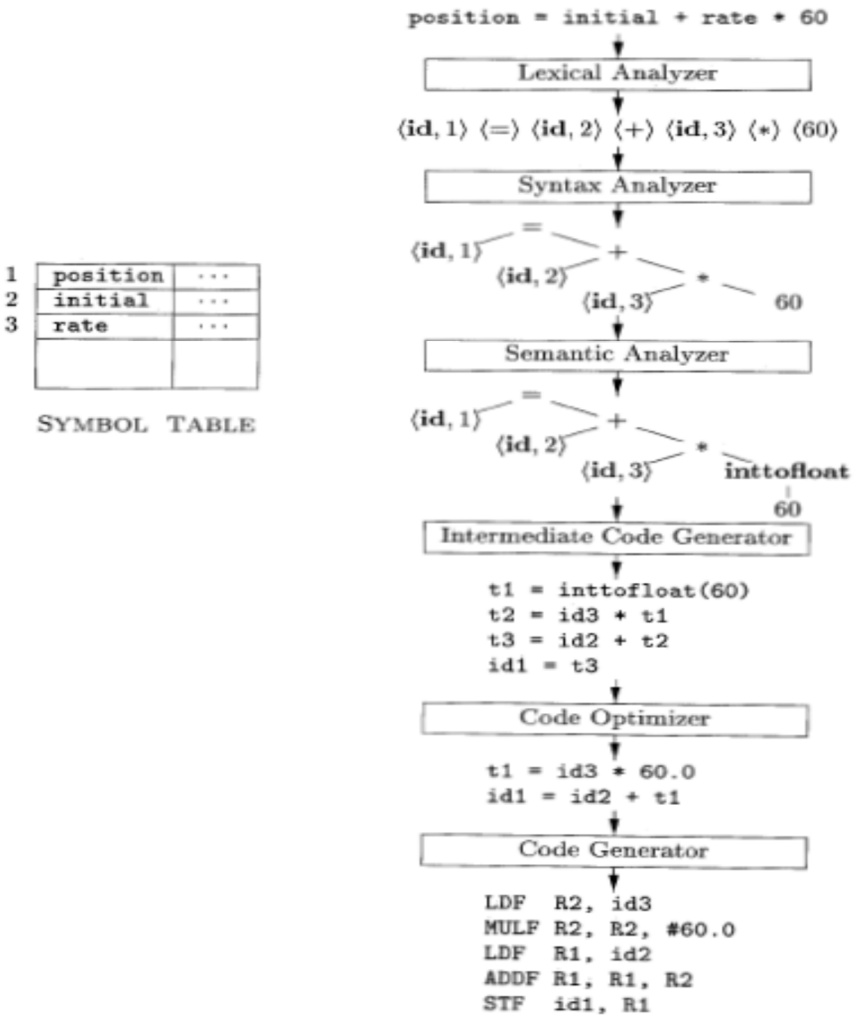

Figure 1.2: Exemples de différentes étapes composant un compilateur