République Algérienne Démocratique et Populaire

Ministère de l 'Enseignement Supérieur et de la Recherche Scientifique

Faculté des Sciences et de l'Ingénierie

Département de Génie Informatique

PROJET DE FIN D'ETUDES

Pour l'obtention du diplôme

D'ingénieur d'état en informatique

Option : Systèmes Parallèles et Distribués

Thème

Réalisation d'un compilateur Source à Source d'un mini langage de programmation arabe vers Pascal

Réalisé par :

BOUSBAA Fatima el zahraa

RAKIK Asma

Encadré par :

Mr. ALLAOUI Tahar

N° d'ordre :……/2008-PFE/DGI

RÉSUMÉ

La conception et la réalisation des langages de programmation, depuis Fortran et Cobol jusqu'au Caml et Java, est l'une des clés du développement et de la maîtrise de systèmes informatiques.

Ce mémoire présente la réalisation d'un compilateur qui fait la traduction d'un programme écrit dans un langage de programmation arabe vers un programme équivalent écrit dans le langage de programmation Pascal.

Notre travail est divisé en deux parties, dans la première, nous introduisons d'abord la théorie des langages de programmation et les concepts de base de la compilation.

Dans la deuxième partie, nous présentons la conception et la réalisation de notre compilateur avec une interface simple.

Cet outil réalisé va permettre la compréhension des différentes étapes de la compilation et d'apprendre la programmation arabe.

Mots clés : langage de programmation, compilateur.

TABLES DES MATIÈRES

No table of contents entries found.

TABLES DES FIGURES

No table of contents entries found.

INTRODUCTION

L'informatique est devenue indispensable dans tous les domaines, grâce à l'utilisation de l'ordinateur qui rend automatique, rapide et facile la réalisation des différentes tâches. On trouve que l'utilisateur ne peut exprimer ses idées qu'en langage naturel, alors que l'ordinateur ne comprend que le langage machine.

La communication directe entre l'utilisateur et la machine nécessite un langage qui peut être compréhensible par les deux.

Il est évident qu'un tel langage n'existe pas, alors les langages de programmation ont été crées afin de facilité le passage d'un langage compréhensible par l'humain vers un langage compréhensible par la machine.

Les besoins des utilisateurs varient, évoluent, et les langages de programmation aussi, presque chaque jour des nouveaux langages sont crées et des nouvelles fonctionnalités sont ajoutés aux anciens langages de programmation.

Actuellement, les langages de programmation utilisés à grande échelle sont basés sur le lexique et la syntaxe anglaise ou française, on constate que la langue arabe n'a pas encore occupée une place importante dans le domaine de la programmation. Cette absence à une influence négative sur les environnements qui utilisent la langue arabe.

Pour cette raison nous essayons de réaliser un premier pas vers un compilateur qui traduit un programme écrit dans un mini langage de programmation arabe en un programme équivalent dans le langage de programmation Pascal.

L'objectif principal de notre projet est de présenter et d'implémenter les différentes étapes de la compilation, le choix de la langue arabe a pour but d'encourager les informaticiens à exploité cette langue dans leurs systèmes d'information.

Ce mémoire est composé de quatre chapitres, le premier consiste à présenter l'évolution des langages de programmation depuis le premier langage jusqu'au nos jours.

Ensuite, on passe au deuxième chapitre : nous introduisons les concepts de base de la compilation qui permet de mieux comprendre notre travail.

Le troisième chapitre sera consacré à l'étude conceptuelle du notre compilateur : nous expliquons les différentes méthodes et outils de réalisation de notre compilateur.

Le dernier chapitre comprend la description détaillée de l'application développée. Un exemple d'exécution est donné pour illustrer les différentes caractéristiques de l'application.

Nous terminons par une conclusion qui contient les résultats obtenus et les améliorations futures de notre travail.

A la fin de ce mémoire, deux annexes sont mises à votre disposition :

La première annexe représente les structures de données utilisées et les différentes procédures appliquées.

La deuxième annexe contient la structure générale de notre mini langage de programmation qui permet de bien maîtriser notre langage.

CHAPITRE 1

LES LANGAGES DE PROGRAMMATION

Un langage de programmation est un code de communication permettant à un être humain de dialoguer avec une machine.

Les langages de programmation ont été crées avec l'apparition des ordinateurs, ils ont été fondus essentiellement pour effectuer les grands calculs.

Dans ce chapitre nous présentons l'historique d'évolution des langages de programmation, et nous citons quelques langages de programmation arabes.

1.1 Introduction

Tout a commencé dans les années 1950 avec le langage Fortran, Cobol et Lisp, même s'il y a eu de nombreux précurseurs. Ainsi les grecs, les chinois savaient calculer, automatiser (à la main) des calculs. Nous ne retiendrons donc que deux dates importantes. En 820, le mathématicien El Khawarizmi a publié à Bagdad un traité intitulé "La science de l'élimination et de la réduction" qui, importé en Europe occidentale lors des invasions arabes a eu une grande influence sur le développement des mathématiques. En 1840, Ada Lovelace a défini le principe des itérations successives dans l'exécution d'une opération. En l'honneur d'El Khawarizmi elle a nommé "Algorithme" le processus logique d'exécution d'un programme. [S1]

1.2 Définition des langages de programmation

Selon Jerzy [Jerz.98], les langages de programmation permettent de définir les ensembles d'instructions effectuées par l'ordinateur lors de l'exécution d'un programme. Il existe des milliers des langages de programmation, la plupart d'entre eux étant réservés à des domaines spécialisés. Ils font l'objet de recherches constantes dans les universités et dans l'industrie.

1.3 Historique des langages de programmation

L'évolution des langages de programmation est passée par plusieurs générations, chaque génération a permis de développer la génération suivante.

Les années 50

En 1950, l'invention de l'assembleur par Maurice V. Wilkes de l'université de Cambridge ouvre la voie aux langages dits "de haut niveau". Avant, la programmation s'effectuait directement en binaire.

Grace Murray Hopper a développé le premier compilateur, nommé A0. Il permet de générer un programme binaire à partir d'un "code source".

Alors que le langage Fortran commence à apparaitre vers 1955 notamment chez IBM, (International Busines Machine).

Les années 60

En 1964, Thomas Kurtz et John Kemeny ont crée un langage pour "débutants" se nomme Basic.

Les années 70

En 1968, Niklaus Wirth a développé Algol, il a crée un successeur d'Algol nommé le langage Pascal. Ce langage, moins puissant qu'Algol, est bien structuré, très lisible et se trouve donc bien adapté à l'enseignement de la programmation. Il va être pendant une quinzaine d'années le langage préféré de nombreux enseignants en informatique.

En 1978, Brian Kernighan a écrit un livre au sujet de la programmation en langage C.

Les années 80

En 1983 Bjarn Stroustrup a développé une extension orientée objet au langage C qui deviendra le langage C++.

Les années 90

En 1995, Java est introduit comme langage de développement objet.

Les années 2000

Les années 2000 ne voient pas apparaitre de nouveau langage marquant. La tendance est plutôt à l'amélioration et à l'enrichissement des anciens langages.

Quelques exemples de progression de version :

	2000	2001	2002	...	2005	2006
Php	4.0	4.1	4.2		5.0.5	5.1
Java	1.3		1.4		1.5	2.1

De même Delphi, le successeur de Pascal, Turbo Pascal et Pascal Windows propose une interface complète de développement sous Windows et Kylix sous Linux. [S1]

Comme nous remarquons, il existe des milliers de langages de programmation, la majorité des langages utilisent la langue anglaise ou française pour définir ses structures, bien que la langue arabe n'a pas encore prouvé sa place dans le domaine de programmation, quelques langages de programmation arabes ont été construits, ces langages restent toujours dans un cadre académique, ils n'ont pas été commercialisé à grande échelle.

1. 4 Les langages de programmation arabes

En 1978 : l'université de Mawsil (Iraq) a crée le langage de programmation arabe qui a été nommé ' غريب', ses caractéristiques:

- conforme au langage Basic.
- son compilateur en Fortran.
- fait des calculs arithmétiques.
- le curseur de droite à gauche.

L'université militaire iraquienne a crée un langage de programmation arabe pour les débutants 'الخوارزمي' dans la même année (1978).

En 1979 la société Saoudite Otram a traduit le langage Basic en arabe, ce langage nommé ' سلطانة'.

En même temps un autre langage nommé aussi 'الخوارزمي' a été développé en Californie.

En 1980-1981 l'université de Kuwait a développé un traducteur du langage Basic qui a été nommé 'صخر' avec une autorisation de Microsoft.

En 1984 l'université du Roi Abed Elaziz - Arabie Saoudite - a crée un langage qui rassemble les propriétés des langages (C, Basic, Pascal) qui a été nommé 'ضاد'.

En 1986 le docteur Alafendi en université de Khourtoum- Soudan- fait une traduction de langage Pascal sous le nom 'سينا'.

En 1988 le docteur Hassen Madhkour et Ahmed Mahdjoub à l'université de l'Arabie saoudite ont crée le langage 'باسكال العربي'.

En 1994 le docteur Alafendi à l université de l'Arabie Saoudite a crée le langage 'سنبلة'.

En 1996 le docteur Abed Elmalek Elsalmen a développé le langage 'باسكال العربي'.

En 2006 le docteur Syrien Mohamed Amar Salka a crée un langage traduit le langage C qui a été nommé 'ج'. [S2]

Nous remarquons que le nombre des langages de programmation arabes est très limité à cause des problèmes que rencontrent le développement de ces langages, parmi les causes on distingue :

- Les différents codages des lettres arabes.
- Les problèmes du financement.
- La faiblesse de la maintenance et de la mise à jour.
- L'existence des langages standards et plus puissants (Java, C++, Pascal..).
- L'absence de la communication entre les développeurs arabes.

Malgré ces problèmes, il existe toujours des essais pour créer des nouveaux langages de programmation arabes.

1. 5 Conclusion

Plusieurs langages de programmation ont été crées, et les programmes écrits en ces langages sont toujours plus proches du langage naturel, pour permettre à l'ordinateur de comprendre et exécuter les programmes, on fait appel à un processus nécessaire qui est le processus de compilation.

Dans le chapitre suivant on jette la lumière sur les notions de base de la compilation.

CHAPITRE 2

LA COMPILATION :

CONCEPTS DE BASE

*Jour après jour, tout programmeur utilise un outil essentiel à la réalisation de ses besoins : **un programme**, Un programme est la manifestation d'une pensée qui s'exprime sous la forme d'un texte. Ce texte désigne des opérations qui doivent être réalisées par une machine qui exécute ainsi le texte. Un texte s'exprime dans un langage. Si ce langage est compris par la machine, cela signifie que la machine peut exécuter directement le texte qui lui est proposé. Dans le cas contraire, il est nécessaire de traduire le texte source, en un texte acceptable par la machine, le texte cible. L'opération conduisant à cette traduction s'appelle compilation.*

Dans ce chapitre, nous présentons une vue générale sur la compilation, nous définissons aussi les concepts de base de la compilation, qui sont primordiaux pour la suite de ce mémoire.

2.1Introduction

La notion de la compilation est apparue en informatique à la fin des années 50 avec le langage Fortran. Avant Fortran, les programmes ont été écrits directement en langage machine, c.-à-d. des suites de 0 et 1, ces programmes sont incompréhensibles par l'humain et il est difficile de détecter les erreurs. Les programmeurs ont essayé de faciliter la programmation.

En premier lieu ils donnent des symboles aux instructions machines, ainsi, le langage d'assemblage apparait, par exemple, on écrivait MOV R0, R1 au lieu de 0A 43.

Malgré les avantages non contestés du langage d'assemblage, il présente des inconvénients :

- La manipulation des programmes très longs.
- Les caractéristiques de la machine doivent être connues.
- Les erreurs sont difficiles à corriger.

Les inconvénients des langages machines et des langages d'assemblage obligent les concepteurs des langages à s'affranchir complètement de la connaissance de la machine en élaborant des langages de haut niveau, ces langages évolués permettent d'utiliser:

- ✓ Expressions arithmétiques.
- ✓ Variable locale.

- ✓ Fonction avec paramètres.
- ✓ Structure de données : tableau, structure, liste…

Les langages évolués ne sont pas proches du langage machine, pour cela il a fallu construire des programmes qui traduisent les énoncés exprimés dans le langage de haut niveau en langage machine. Les programmes effectuant ce genre d'opération s'appellent des compilateurs et l'opération s'appelle compilation.

2.2 Définitions

2.2.1 Compilation

Selon Garreta [Gar.01], dans le sens le plus usuel du terme, la compilation est une transformation que l'on fait subir à un programme écrit dans un langage évolué pour le rendre exécutable. Fondamentalement, c'est une traduction : un texte écrit en 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.

2.2.2 Traduction

Est une transformation d'un programme écrit dans un langage source en un programme équivalent écrit dans un langage cible.

2.2.3 Traducteur

Selon Paulin [Pau04], un traducteur est un programme qui transforme chaque mot d'un langage L1 « langage source » sur un alphabet A1 en un mot d'un langage L2 « langage cible » sur un alphabet A2.

Figure 2.1 : un traducteur

On distingue différents types de traducteurs :

a) Assembleur

Le langage source des assembleurs utilise une représentation symbolique des instructions machines. Le langage cible est alors le langage machine. Un assembleur traduit généralement une instruction symbolique en une instruction machine.

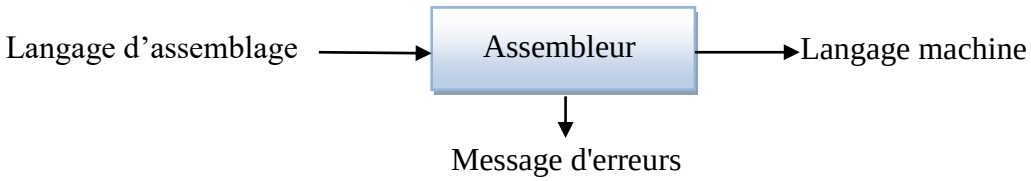


Figure 2.2 : un assembleur

b) Compilateur

A.Aho [Aho.89] dit qu'un compilateur est un logiciel particulier qui lit un programme écrit dans un langage de haut niveau « langage source » et le traduit en un programme équivalent écrit dans un langage machine « langage cible » (voir la figure 2.3) au cours de ce processus de traduction, un rôle important du compilateur est de signaler la présence des éventuelles erreurs dans le programme source.

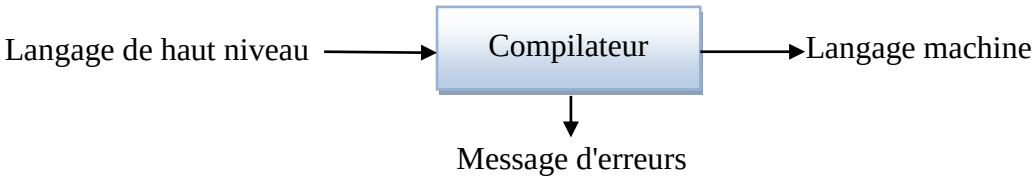


Figure 2.3 : un compilateur

c) Interpréteur

Au lieu de produire un programme cible comme dans le cas d'un compilateur, un interprète exécute lui-même au fur et à mesure les opérations spécifiées par le programme source, il analyse une instruction après l'autre puis l'exécute immédiatement. [S3]

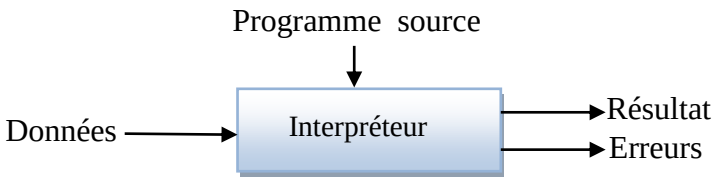


Figure 2.4 : un interpréteur

2.3 Structure d'un compilateur

Un compilateur fonctionne en deux étapes, l'analyse et la synthèse (voir la figure 2.5), c'est-à-dire qu'au lieu de remplacer chaque construction du langage source par une suite équivalente de constructions du langage cible, il commence par analyser le texte source pour en construire une représentation intermédiaire qu'il traduit à son tour en langage cible.

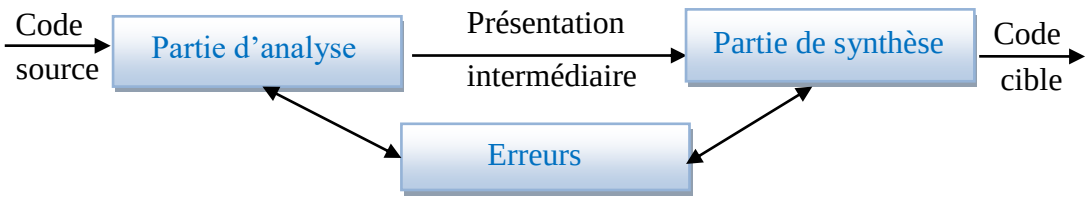


Figure 2.5: parties de compilation

2.3.1 La phase d'analyse

La phase d'analyse permet de partitionner le programme en ses constituants et de créer une représentation intermédiaire, elle est indépendante du code cible. Cette analyse est devisée en trois parties (l'analyse lexicale ; l'analyse syntaxique ; l'analyse sémantique). [Aho.89][S4]

a) L'analyse lexicale

L'analyseur lexical constitue la première étape d'un compilateur. Sa tâche principale est de lire les caractères d'entrée et de produire comme résultat une suite d'unités lexicales que l'analyseur syntaxique va traiter. En plus, l'analyseur lexical réalise certaines tâches secondaires comme l'élimination des caractères superflus (commentaires, tabulations, fin de lignes, ...), et gère aussi les numéros de ligne dans le programme source pour pouvoir associer à chaque erreur rencontrée par la suite la ligne dans laquelle elle intervient.

b) L'analyse syntaxique

Chaque langage de programmation à un ensemble des règles appelées les règles de production. Un programme est dit syntaxiquement correct s'il respecte ces règles.

Le principe est d'essayer de construire un arbre de dérivation. Il existe deux méthodes pour cette construction : analyse descendante et analyse ascendante.

Dans l'analyse descendante on essaye de construire l'arbre de dérivation à partir de l'axiome vers les unités lexicales. D'une façon opposée, l'analyse ascendante établit des réductions sur la chaîne à analyser pour atteindre l'axiome de la grammaire.

c) L'analyse sémantique

Les analyseurs lexicaux et syntaxiques ne sont pas compétents à assurer la correction de certaines propriétés du langage. Par exemple:

- On ne peut pas utiliser dans une instruction une variable qui n'a pas été déclarée au préalable.
- On ne peut pas déclarer deux fois une même variable.
- On ne peut pas multiplier un réel avec une chaîne.
- …

Le rôle de l'analyse sémantique est donc de vérifier ces contraintes. Elle se fait en général en même temps que l'analyse syntaxique, à l'aide d'actions sémantiques insérées dans les règles de productions.

2.3.2 La phase de synthèse

Cette phase permet de créer un programme cible à partir de la représentation intermédiaire du programme source, dans ce cas on distingue les étapes suivantes [Aho.89]

a) La génération du code intermédiaire

La génération du code intermédiaire consiste à créer un programme équivalent au programme source en un langage plus proche du langage cible, cette étape doit être facile à produire et facile à comprendre par la machine.

b) L'optimisation du code intermédiaire

Cette étape permet d'améliorer le code produit de telle sorte que le programme résultant soit plus rapide. Quelques optimisations sont triviales, par exemple : l'élimination des variables intermédiaire, et d'autres sont complexes, par exemple : définir des tâches en parallèle.

c) La génération du code

C'est la dernière étape de la compilation, cette étape produit un code final équivalent au code du départ « code source » soit en assembleur, soit en langage machine.

2.3.3 La détection des erreurs

C'est une partie importante de compilation, une bonne gestion des erreurs permet de détecter un maximum d'erreurs, et de déterminer le plus précisément possible les causes de ces erreurs. Après avoir détecté une erreur, il s'agit ensuite de la traiter de telle manière que la compilation puisse continuer et que d'autres erreurs puissent être détectées.

2.3.4 La gestion de la table des symboles

La table des symboles est une structure de données servant à stocker des informations concernant le contenu d'un programme source (par exemple : le type d'identificateur, leur emplacement mémoire, ...). Cette table est utilisée tout au long de la compilation depuis son remplissage par l'analyseur lexical jusqu'à la génération du code.

La figure suivante décrit sommairement les différentes étapes de la compilation.

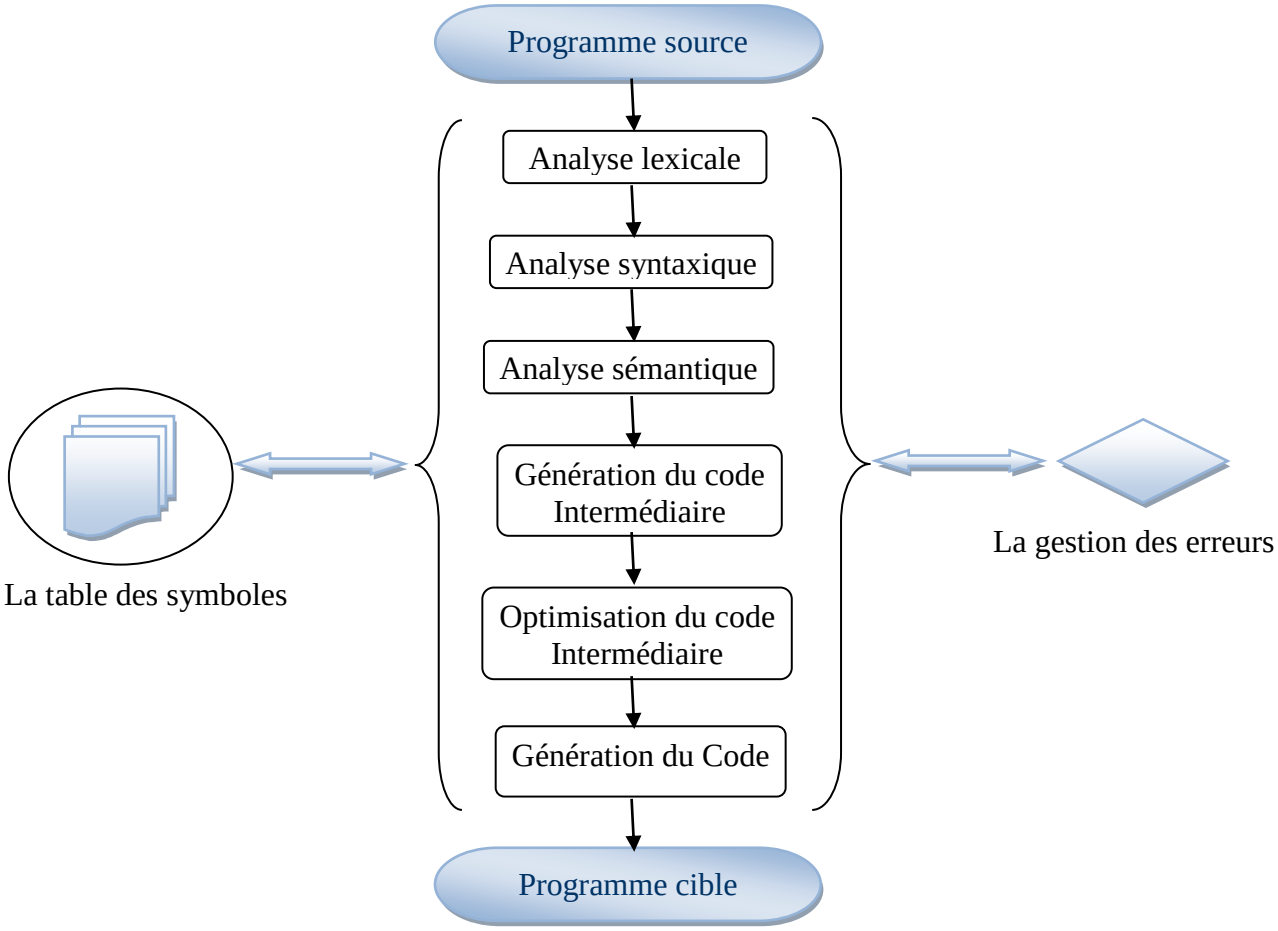


Figure 2.6: la structure d'un compilateur

2.4 Conclusion

Dans ce chapitre nous avons défini la compilation comme un processus de transformation de texte (texte source et texte cible), nous avons présentées les principales phases de la construction d'un compilateur, c'est-à-dire un programme qui transforme une suite de caractères représentant un programme en une suite d'instructions machine qui pourront s'exécuter pour produire le résultat du programme.

Dans le chapitre suivant on va présenter l'étude conceptuelle de notre compilateur avec les différentes techniques utilisées.

CHAPITRE 3

ÉTUDE CONCEPTUELLE

DE NOTRE COMPILATEUR

La compilation n'est pas limitée à la traduction d'un programme écrit dans un langage de programmation de haut niveau en un programme directement exécutable par une machine, cela peut aussi être :

La traduction d'un programme écrit en langage de programmation de haut niveau « langage source » vers un autre programme écrit en un autre langage de programmation de haut niveau « langage source » par exemple : une traduction de Pascal vers C++, ou de Java vers C++.

Dans ce chapitre nous présentons notre compilateur qui traduit un programme écrit dans un mini langage de programmation arabe vers le langage de programmation Pascal.

Le but de ce chapitre est de présenter les principes de base de la conception de notre compilateur.

3.1Introduction

Après avoir présenté les notions de base de la compilation dans le chapitre précédent, nous exposons dans ce qui suit les étapes de réalisation de notre compilateur, tout en montrant la structure détaillée de notre mini langage.

3.2 Les étapes de réalisation

a) Idée de base

Nous avons eu la chance d'étudier les notions de base de la compilation l'année précédente. Cette occasion, nous a motivé à exploiter nos connaissances pour enrichir le domaine de programmation par un mini langage arabe.

Nous avons montré que la réalisation d'un langage de programmation de l'analyse lexicale à la génération du code final est une tâche très difficile qui dépasse le cadre de notre

travail, à cet effet, nous choisissons de réaliser un compilateur qui traduit un programme écrit dans le mini langage qu'on a développé vers un programme en langage Pascal.

b) Pourquoi le langage Pascal ?

Pascal est le langage le plus adapté à l'enseignement du à sa structure claire et puisque notre but est de réaliser un mini langage simple qui peut être utilisé dans le cadre éducatif, nous avons choisi Pascal comme résultat de traduction.

c) Structure générale de notre mini langage

La structure de notre mini langage est proche de celle de Pascal.

Elle permet d'écrire des programmes constitués d'une entête, partie de déclarations, et partie d'actions.

On peut réaliser des opérations arithmétiques, les instructions de test et des instructions répétitifs.

Notre mini langage permet de faire appel aux sous programmes afin de réduire la taille des programmes écrits.

3.3 Mise en œuvre de L'analyseur lexical

3.3.1Introduction

Comme on a pu le voir dans le chapitre précédent le rôle principal d'un analyseur lexical est de lire le programme source 'caractère par caractère', de connaitre les mots du programme et de remplacer chaque mot par une 'unité lexicale'. L'unité lexicale peut être un identificateur, un mot clé, une constante, un opérateur ou un séparateur.

3.3.2 Les unités lexicales

a) Les identificateurs

Mots formés par l'utilisateur pour l'appellation des variables et des constantes de son programme.

Les identificateurs de notre langage doit commencer par une lettre, suivie par des lettres ou des chiffres.

Un nom de programme respecte les règles liées aux identificateurs.

b) Les mots clés

Mot particulier du langage ayant un sens bien spécifique.

Les mots clés de notre mini langage sont classés comme suit :

partie du programme	Mots clés	Le Rôle des mots clés
La partie d'entête	برنامج	Notre programme commence par برنامج
La partie de déclarations	ثوابت	Déclarer les constants
	متغيرات	Déclarer les variables
	صحيح , حقيقي , سلسلة , حرف , من , جدول	Définir les types des variables qui sont déclarées
La partie d'actions	إبدأ , إنتهى	Déterminer un ensemble d'actions
	إذاكان فإن وإلا	Instruction conditionnelle
	مادام . . . قم أعد . . . حتى لأجل . . . قم	Instructions répétitifs
	إقرأ , إقرأ_سطر	La lecture des variables
	أكتب , أكتب_سطر	L'affichage des variables ou des mots
	و , أو	Opérateurs logiques
	وظيفة , إجراء	Déclarer les procédures et les fonctions
	مناداة_وظيفة , مناداة_إجراء	Appeler les procédures et les fonctions

c) Les opérateurs arithmétiques + - * /

d) Les opérateurs relationnels <= >= < > = %

e) Les séparateurs . ، ؛ ( ) :

3.3.3 Les commentaires dans notre code :

Nos commentaires peuvent tenir sur une seule ligne (exemple : ici nos commentaires \ ) comme sur plusieurs (exemple : {ici nos commentaires}).

Une fonction importante de cette étape est d'installer tous les identificateurs et touts les mots clés ainsi que les constants du programme dans la table des symboles .toutes les autres étapes de la compilation sont en relation permanente avec cette table.

3.3.4 Les automates et les expressions régulières

a) Identificateur

id ⟶ Lettre (Lettre / Chiffre)*

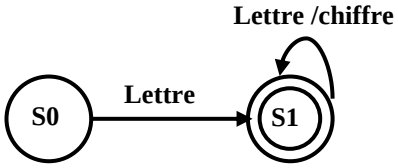


Figure3.1 : Automate de l'identificateur

b) Nombre

Nombre ⟶ Chiffre (Chiffre)* . Chiffre (Chiffre)*

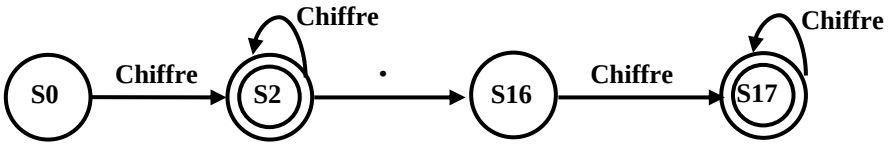


Figure3.2 : Automate d'un nombre

c) Opérateur fin

Op_Fin ⟶ .

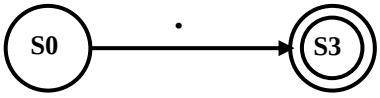


Figure3.3 : Automate d'opérateur fin

d) Séparateur d'instruction

Sep_Inst ⟶ ;

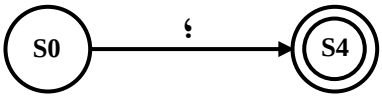


Figure3.4 : Automate de séparateur d'instruction

e) Parenthèse ouvrant

P_O ⟶)

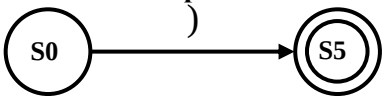


Figure3.5 : Automate de parenthèse ouvrant

f) Parenthèse fermant

P_F ⟶ (

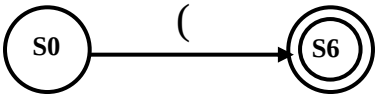


Figure3.6 : Automate de parenthèse fermant

g) Séparateur variable

Sep_Var ⟶ ‘

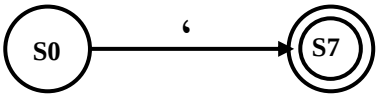


Figure3.7 : Automate de séparateur variable

h) Opérateur de multiplication et de division

Op_Mul ⟶ */ /

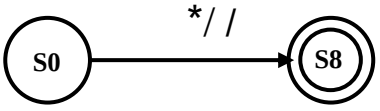
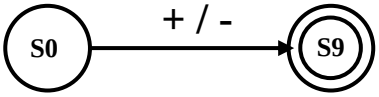


Figure3.8 : Automate de multiplication et de division

i) Opérateurs d'addition et de soustraction



Op_Add → + / -

Figure3.9 : Automate d'addition et de soustraction

j) Affection

Affectation → :=

Figure3.10 : Automate d'affectation

k) Opérateurs relationnels

Op_Rel → (< / >) =

Figure3.11 : Automate d'opérateurs relationnels

Op_Rel → % / =

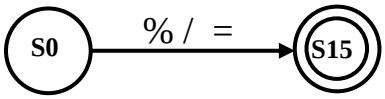


Figure3.12 : Automate d'opérateurs relationnels

l) Opérateur de chaine

Op_Chaine → "

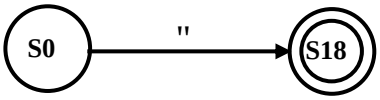


Figure3.13 : Automate d'opérateur de chaine

m) Indice de Table

Indice1_Tab →[

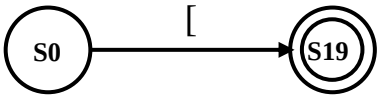


Figure3.14: Automate d'indice de table

Indice2_Tab →]

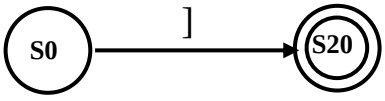
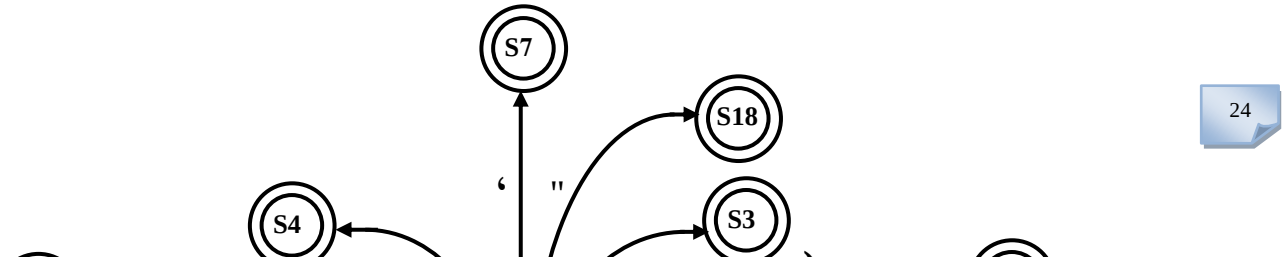


Figure3.15: Automate d'indice de table

3.3.5 L'automate union déterministe du langage



Chiffre

Figure3.16: Automate union déterministe du langage

3.3.6 La détection des erreurs lexicales

Les erreurs lexicales se produisent lorsque l'analyseur est confronté à une suite de caractères qui ne correspond à aucun des modèles d'unité lexicale.

L'analyseur lexical détecte les erreurs suivantes :

- ✓ Pas de texte source: si l'éditeur de texte est vide.
- ✓ Caractère illégal: si l'analyseur rencontre par exemple la chaine 'aبb', il peut juste signaler l'erreur : « المتغير 'aبb' غير مقبول لأن الحروف 'ab' لا تنتمي إلى حروف اللغة »
- ✓ Identificateurs erroné: si l'analyseur rencontre par exemple la chaine 'أب1', il signal l'erreur : « المتغير '1أب' غير مقبول لأن المتغير لا يبدأ بعدد »
- ✓ Identificateurs très long: lorsque la taille du mot dépasse 65 caractères.

Lorsque l'analyseur est confronté a une suite des caractères ne correspond a aucun de ses modèles, il utilise la stratégie du mode panique qui permet d' ignorer l'unité lexicale qui pose problème, de signaler l'erreur et de continuer [Sup04].

3.3.7 Résultat d'analyse lexicale

A la fin de la première partie de la compilation, l'analyseur lexical génère une suite d'unités lexicales à partir du programme source, ces unités lexicales représentent l'entrée de la prochaine analyse.

3.4 Mise en œuvre de L'analyseur syntaxique

3.4 .1 Introduction

L'analyseur syntaxique prend comme entrée la suite des unités lexicales et doit reconnaitre si cette entrée appartient au langage utilisé, en d'autre terme, l'analyseur syntaxique doit vérifier que cette suite respecte la syntaxe « la grammaire » du langage.

3.4.2 La grammaire de notre mini langage

1) Programme ⟶ برنامج id ؛ PRG .
2) PRG ⟶ Partie_Dec_Const Partie_Dec_Var PRG_Proc_Fct Bloc_Inst
3) Partie_Dec_Const ⟶ ثوابت id = Const ؛ Dec_Const
4) Partie_Dec_Const ⟶ ξ
5) Dec_Const ⟶ id = Const ؛ Dec_Const
6) Dec_Const ⟶ ξ
7) Const ⟶ Signe Nombre
8) Const ⟶ "Chaine"
9) Signe ⟶ Op_Add
10) Signe ⟶ ξ

11) Partie_Dec_Va r متغيرات id V : Types Type ؛ Dec_Var
12) Partie_Dec_Var ع
13) Dec_Var id V : Types Type ؛ Dec_Var
14) Dec_Var ع
15) V ، id V
16) V ع
17) Types ⟶ جدول [Signe Nombre . . Signe Nombre Dimension] من
18) Types ⟶ ع
19) Dimension ⟶ ، Signe Nombre . . Signe Nombre Dimension
20) Dimension ⟶ ع
21) Type ⟶ صحيح
22) Type ⟶ حقيقي
23) Type ⟶ سلسلة
24) Type ⟶ حرف
25) PRG_Proc_Fct ⟶ PRG_Proc PRG_Proc_Fct
26) PRG_Proc_Fct ⟶ ع
27) PRG_Proc ⟶ Procédure
28) PRG_Proc ⟶ Fonction
29) Procédure ⟶ إجراء id Dec_Proc ؛ PRG ؛
30) Dec_Proc ⟶) Dec_Par id V : Types Type Dec_Var_Proc (
31) Dec_Proc ⟶ ع
32) Dec_Par ⟶ متغيرات
33) Dec_Par ⟶ ع
34) Fonction ⟶ وظيفة id Dec_Proc : Type ؛ PRG ؛
35) Bloc_Inst ⟶ إبدأ Instructions إنتهى
36) Instructions ⟶ Inst ؛ Instructions
37) Instructions ⟶ ع
38) Inst ⟶ Lecture
39) Inst ⟶ Ecriture
40) Inst ⟶ Affectation
41) Inst ⟶ Inst_Conditionnelle
42) Inst ⟶ Boucle_TQ
43) Inst ⟶ Boucle_PR
44) Inst ⟶ Boucle_RT
45) Inst ⟶ Ap_Proc

⟶

⟶

⟶

46) Lecture ⟶ Lec **)** Variable var_tab **(**
47) Lec ⟶ **إقرأ**
48) Lec ⟶ **إقرأ_سطر**
49) Ecriture ⟶ Ecr **)** Mot Mots **(**
50) Ecr ⟶ **أكتب**
51) Ecr ⟶ **أكتب _سطر**
52) Mot ⟶ "Chaine"
53) Mot ⟶ Variable
54) Mots ⟶ **،** Mot Mots
55) Mots ⟶ **ع**
56) Affectation ⟶ Variable := Simple_Expression
57) Variable ⟶ id Indice_Tab
58) Indice_Tab ⟶] id_Nbr [
59) Indice_Tab ⟶ **ع**
60) id_Nbr ⟶ id
61) id_Nbr ⟶ Nombre
62) Affectation ⟶ id **:=** Simple_Expression
63) Simple_Expression ⟶ Signe Terme Op_Terme
64) Terme ⟶ Facteur Op_Fact
65) Facteur ⟶ Variable
66) Facteur ⟶ Nombre
67) Facteur ⟶ "Chaine"
68) Facteur ⟶ **)** Simple_Expression **(**
69) Op_Fact ⟶ Op_Mul Facteur Op_Fact
70) Op_Fact ⟶ **ع**
71) Op_Terme ⟶ Op_Add Terme Op_Terme
72) Op_Terme ⟶ **ع**
73) Inst_Conditionnelle ⟶ **إذا كان** Expression **فإن** Bloc_Inst Instruction_Cond
74) Instruction_ Cond ⟶ **وإلا** Bloc_Inst
75) Instruction_ Cond ⟶ **ع**
76) Inst_Cond ⟶ Op_log Expression
77) Inst_Cond ⟶ **ع**
78) Expression ⟶ **)** Simple_Expression Op_Rel Simple_Expression **(** Inst_Cond
79) Boucle_TQ ⟶ **مادام** Expression **قم** Bloc_Inst
80) Boucle_PR ⟶ **لأجل** id = id_Nbr **إلى** id_Nbr **قم** Bloc_Inst

81) Boucle_RT ⟶ أعد Instructions حتى Expression
82) Ap_Proc ⟶ id List_Par
83) List_Par ⟶ مناداة_إجراء) id V (/ ع
84) List_Par ⟶ ع
85) Ap_Fct ⟶ مناداة_وظيفة id) id V (
86) Dec_Var_Proc ⟶ ؛ id V : Types Type Dec_Var_Proc
87) Dec_Var_Proc ⟶ ع
88) Var_Tab ⟶ ، Variable Var_Tab
89) Var_Tab ⟶ ع
90) Facteur ⟶ Ap_Fct

Il existe plusieurs méthodes d'analyse appartenant à l'une des deux catégories d'analyse qui sont l'analyse descendante et l'analyse ascendante.

Dans l'analyse descendante on essaye de construire l'arbre de dérivation à partir de l'axiome vers les unités lexicales. Deux méthodes existent pour cette analyse, l'analyse prédictive LL et l'analyse récursive. D'une façon opposée, l'analyse ascendante établit des réductions sur la chaîne à analyser pour aboutir à l'axiome de la grammaire. Comme pour l'analyse descendante, il existe des méthodes telles que : la méthode SLR, la méthode LR, la méthode LALR.

Notre grammaire respecte les conditions suivantes:

- ✓ N'est pas récursive à gauche ;
- ✓ Est factorisée à gauche ; Alors on conclue que notre grammaire est LL(1).

3.4.3 L'analyse syntaxique prédictive LL

L'analyse prédictive LL utilise une table appelée table prédictive. Les lignes de cette table sont indicées par les symboles non terminaux du langage, par contre, les colonnes de la table indicées par les symboles terminaux [Dria.88],

Les cases de la table contiennent éventuellement des règles de la grammaire. Pour la construction de cette table, on a besoin de créer l'ensemble premier et suivant.

3.4.4 La table de premier et suivant

Non Terminaux	Premier	Suivant
Programme	برنامج	#

PRG	إبدأ إجراء متغيرات ثوابت وظيفة	. ؛
Partie_Dec_Const	ثوابت ع	إبدأ وظيفة متغيرات إجراء
Dec_Const	id ع	إبدأ وظيفة متغيرات إجراء
Const	Op_Add Nombre "	؛
Signe	Op_Add ع	) id Nombre
Partie_Dec_Var	متغيرات ع	إبدأ وظيفة إجراء
Dec_Var	id ع	إبدأ وظيفة إجراء
V	، ع	: (
Types	جدول ع	صحيح حقيقي حرف سلسلة
Dimension	، ع	[
Type	صحيح حقيقي حرف سلسلة	؛ (
PRG_Proc_Fct	وظيفة إجراء ع	إبدأ
PRG_Proc	وظيفة إجراء	إبدأ وظيفة إجراء
Procédure	إجراء	إبدأ وظيفة جراء
Dec_Proc	) ع	؛ :
Dec_Par	متغيرات ع	id
Fonction	وظيفة	إبدأ وظيفة جراء
Bloc_Inst	إبدأ	. ؛ وإلا
Instructions	أعد لأجل مادام إذا كان id ع إقرأ_سطر إقرأ أكتب أكتب_سطر مناداة_إجراء مناداة_وظيفة	إنتهى حتى
Non Terminaux	**Premier**	**Suivant**
Inst	أعد لأجل مادام إذا كان id إقرأ_سطر إقرأ أكتب أكتب_سطر مناداة_إجراء مناداة_وظيفة	؛
Lecture	إقرأ_سطر,إقرأ	؛
Lec	إقرأ_سطر,إقرأ	)
Ecriture	أكتب أكتب_سطر	؛
Ecr	أكتب أكتب_سطر	)
Mot	" id	(،
Mots	، ع	(
Affectation	id	؛

Variable	id	:= ؛ Op_add Op_rel Op_mul (،
Indice_Tab	] ع	:= ؛ Op_add Op_rel Op_mul (
id_Nbr	Id Nombre	[
Simple_Expression	Op_Add " Nombre id)	؛ Op_rel (
Terme	Nombre id) "	؛ Op_add Op_rel (
Facteur	Nombre id) "	؛ Op_add Op_rel Op_mul (
Op_Fact	Op_mul ع	؛ Op_add Op_rel (
Op_Terme	Op_add ع	؛ Op_rel (
Inst_Conditionnelle	إذا كان	؛
Instruction_ Cond	ع والا	؛
Expression	)	؛ قم فإن
Inst_Cond	ع O p_logique	؛ قم فإن
Boucle_TQ	مادام	؛
Boucle_PR	لأجل	؛
Boucle_RT	أعد	؛
Ap_Proc	مناداة_إجراء	؛
List_Par	) ع	؛
Ap_Fct	مناداة_وظيفة	؛
Dec_Var_Proc	؛ ع	(
Var_Tab	، ع	(

3.4.5 La table d'analyse

	برنامج	id	؛	.	ثوابت	=:	متغيرات	:	صحيح	،
Programme	1									
PRG					2		2			
Partie_Dec_Const					3		4			
Dec_Const		5					6			
Const										
Signe		10								
Partie_Dec_Var							11			
Dec_Var		13								
V								16		15
Types									18	
Dimension										19
Type									21	
PRG_Proc_Fct										
PRG_Proc										
Procédure										
Dec_Proc										

	Nombre	Op_Add	Op_Mul	إبدأ	إنتهى	إذا كان	فإن	وإلا	قم	مادام
Programme										
PRG				2						
Partie_Dec_Const				4						
Dec_Const				6						
Const	7	7								
Signe	10	9								
Partie_Dec_Var				12						
Dec_Var				14						
V										
Types										
Dimension										
Type										
PRG_Proc_Fct				26						
PRG_Proc										
Procédure										
Dec_Proc				31						
Dec_Par										
Fonction										

	و	)	(	Op_Rel	أو	إقرأ	أكتب	إقرأ_سطر	أكتب_سطر	لأجل
Programme										
PRG										
Partie_Dec_Const										
Dec_Const										
Const										
Signe		10								
Partie_Dec_Var										
Dec_Var										
V										
Types										
Dimension										
Type										
PRG_Proc_Fct										
PRG_Proc										
Procédure										
Dec_Proc		30								
Dec_Par										
Fonction										

	إلى	أعد	حتى	من	جدول	حقيقي	حرف	سلسلة	وظيفة	إجراء
Programme										
PRG									2	2
Partie_Dec_Const									4	4
4Dec_Const									6	6
Const										
Signe										
Partie_Dec_Var									12	12
Dec_Var									14	14
V										
Types					17	18	18	18		
Dimension										
Type						22	24	23		
PRG_Proc_Fct									25	25
PRG_Proc									28	27
Procédure										29
Dec_Proc									31	31
Dec_Par										

	"	]	مناداة_ إجراء	مناداة_ وظيفة
Programme				
PRG				
Partie_Dec_Const				
Dec_Const				
Const	8			
Signe				
Partie_Dec_Var				
Dec_Var				
V				
Types				
Dimension		20		
Type				
PRG_Proc_Fct				
PRG_Proc				
Procédure				
Dec_Proc				

Remarque :

On remarque que les cases de la table sont mono défini, on conclu alors que la grammaire est LL1

3.4.6 La détection des erreurs syntaxiques

Les erreurs syntaxiques sont provoquées généralement par une unité lexicale mal placée, ou par l'oubli d'une ou plusieurs unités lexicales autrement dit, l'origine des erreurs syntaxiques est un programme qui n'est pas conforme â la grammaire du langage.

Exemple :

- ✓ Affectation sans la partie gauche : ؛ =:أ
- ✓ إبدأ sans . إنتهى

Lorsque l'analyseur syntaxique détecte une erreur, il doit :

- ✓ Indiquer la présence de l'erreur d'une manière claire et précise
- ✓ Traiter l'erreur rapidement pour continuer l'analyse.

- ✓ Traiter l'erreur le plus efficacement possible de manière à ne pas en créer de nouvelle « cause d'erreur, possibilité de correction ».

Il existe plusieurs stratégies de récupération sur erreur : mode panique, au niveau de syntagme, production d'erreur, correction globale.

Parmi ces stratégies on utilise la récupération par production d'erreur : si l'on a une idée assez précise des erreurs courantes qui peuvent être rencontrées, on ajoute à la grammaire des règles de production qui engendrent des constructions erronées.

Exemple : pour détecter l'absence d'une '؛', on ajoute la règle :

Erreur ⟶ برنامج id PRG . à cette règle :

Programme ⟶ برنامج id ؛ PRG .

Et pour insérer '؛' manquant à son emplacement on utilise la stratégie de récupération au niveau de syntagme : l'analyseur peut effectuer des corrections locales lors de détection des erreurs.

3.4.7 Résultat de l'analyse syntaxique

Après les deux premières étapes de la compilation le programme à compiler doit être lexicalement et syntaxiquement correct mais peut contenir des erreurs de sens qui sont détectés par la prochaine étape.

3.5 Mise en œuvre de L'analyseur sémantique

3.5.1 Introduction

Les analyseurs lexicaux et grammaticaux ne sont pas aptes à assurer la correction de certain propriétés du langage, par exemple, lorsqu'on rencontre un identificateur, est-ce une fonction ou une variable ; si c'est une variable, quel type de valeur est stockée dans cette variable ; si c'est une fonction, combien et quel type d'argument a-t-elle ?.On doit donc assurer qu'un programme bien formé lexicalement et syntaxiquement a un sens. Donc, l'étape suivante de conception de notre compilateur est l'analyse sémantique dont la partie la plus visible est le contrôle de type. Des exemples de taches liées au contrôle de type sont :

- ✓ Mémoriser les types définis par l'utilisateur, lorsque notre langage le permet,
- ✓ Traiter les déclarations des variables, des fonctions ou des procédures, et mémoriser les types qui leur sont appliqués,
- ✓ Vérifier que toute variable, toute fonction et procédure appelée ont bien été préalablement déclarées,
- ✓ Vérifier que les paramètres des fonctions ont les types requis,
- ✓ Contrôler les types des opérandes des opérations arithmétiques et en déduire le type du résultat (conversion de type),
- ✓ Etc.

3.5.2 La gestion de la table des symboles

Pour notre langage, cela consiste essentiellement à vérifier que les identificateurs utilisés sont bien déclarés. Pour tester la déclaration ou non des identificateurs utilisés, on ajoute une colonne dans la table des symboles pour mémoriser le type de chaque identificateur.

3.5.3 La détection des erreurs sémantiques

L'analyseur sémantique détecte la présence de l'erreur d'une manière claire et précise : emplacement d'erreur (numéro et ligne), sélection de ligne, la cause d'erreur, par exemple :

- ✓ Détecter la déclaration d'une même variable deux fois.
- ✓ Lors d'un appel de fonction, il doit y avoir autant de paramètres formels que de paramètres effectifs, et les types des paramètres effectifs doit correspondre aux paramètres formels.
- ✓ Indiquer l'absence de déclaration d'une variable.
- ✓ …

3.5.4 Résultat d'analyse sémantique

Après la réalisation des trois premières étapes de la compilation, on est sûr que le programme compilé ne contient aucune erreur, on passe à l'étape de la traduction du programme vers le langage de programmation Pascal pour continuer les étapes de compilation.

3.6 L'étape de traduction

Selon Philipe [Phil.99], un traducteur est un programme qui accepte en entrée une représentation textuelle d'un algorithme décrit dans un langage source et qu'il produit en sortie une représentation du même algorithme dans un autre langage appelé langage cible.

Dans notre traducteur le langage source est le langage arabe et le langage cible est le langage Pascal.

Figure 3.17 : un traducteur

✓ **La gestion de la table des symboles**

Lorsqu'on traite la traduction d'un programme on ajoute dans la table des symboles une colonne pour traduire chaque mot écrit dans le mini langage arabe en un mot équivalent dans le langage Pascal.

3.7 Conclusion

Dans ce chapitre nous avons présentés les détails de conception de notre compilateur avec les techniques et les outils utilisés pour développer notre application. Le seul problème qui nous rencontre c'est que le langage de programmation Pascal n'accepte pas les caractères arabes, donc, la solution est : La traduction des identificateurs arabes se fait d'une manière séquentielle en langage Pascal (id1, id2, id3…idn).

Dans le chapitre suivant nous présentons l'application développée dans le cadre de ce projet.

CHAPITRE 4

LA RÉALISATION

DU COMPILATEUR

Dans ce chapitre, nous présentons l'application que nous avons développée qui est le fruit de ce travail et nous expliquons en détails la démarche adoptée pendant sa réalisation, ainsi que des exemples explicatifs.

4.1introduction

La réalisation d'un compilateur qui fait la traduction d'un programme écrit dans un mini langage de programmation arabe vers le langage de programmation Pascal était au démarrage un objectif et maintenant une réalité ce qui présente pour nous une réussite. Cette réussite nous a encouragé à nommer notre compilateur 'Najah'.

4.2 Environnement de travail

Najah a été développé sur une machine de type Intel Pentium IV, d'une horloge de 3200Mhz, et de 256 Mo de SDRAM, sous Windows Xp.

4.3 L'outil de développement

Najah v 0.1est développé sous Delphi 2007 de la société CODEGEAR, une filiale de BORLAND pour gérer les outils de développement.

Borland Delphi est un environnement de programmation visuelle orienté objet permettant de développer des applications 32 bits en vue de leur déploiement sous Windows et sous Linux.

Avec Delphi, nous pouvons créer de puissantes applications avec un minimum de programmation.

Delphi propose un ensemble d'outils de conception pour le développement rapide d'applications (RAD), il gère la programmation orientée objet avec La bibliothèque de classes VCL (Bibliothèque de composants visuels) pour Windows et CLX (La bibliothèque de composants Borland multi plate-forme) pour Windows et Linux. [S5][G.dev].

4.4 Description générale de l'application

Notre application est caractérisée par une interface simple et riche permettant :

- ✓ D'ouvrir et de sauvegarder les fichiers à l'aide des boites de dialogue.
- ✓ De colorer Les mots réservés.
- ✓ D'afficher les structures de données utilisées (table de transition, table de symboles, chaîne de référence, table d'analyse, table de premier et de suivant, l'arbre syntaxique).
- ✓ Gestion des erreurs (nature, sélection de la première erreur, numéro de ligne, possibilité de correction).
- ✓ Commandes standards (couper, copier, coller, impression, recherche, …).
- ✓ Extension particulière (.naj) aux fichiers sources.
- ✓ zones de texte.
- ✓ Menus, aide.
- ✓ Boite d'information.
- ✓ Barre d'état.

4.5 Description détaillée de l'application

➢ Lancer l'installation du Najah à partir du CD. La fenêtre suivante s'affiche :

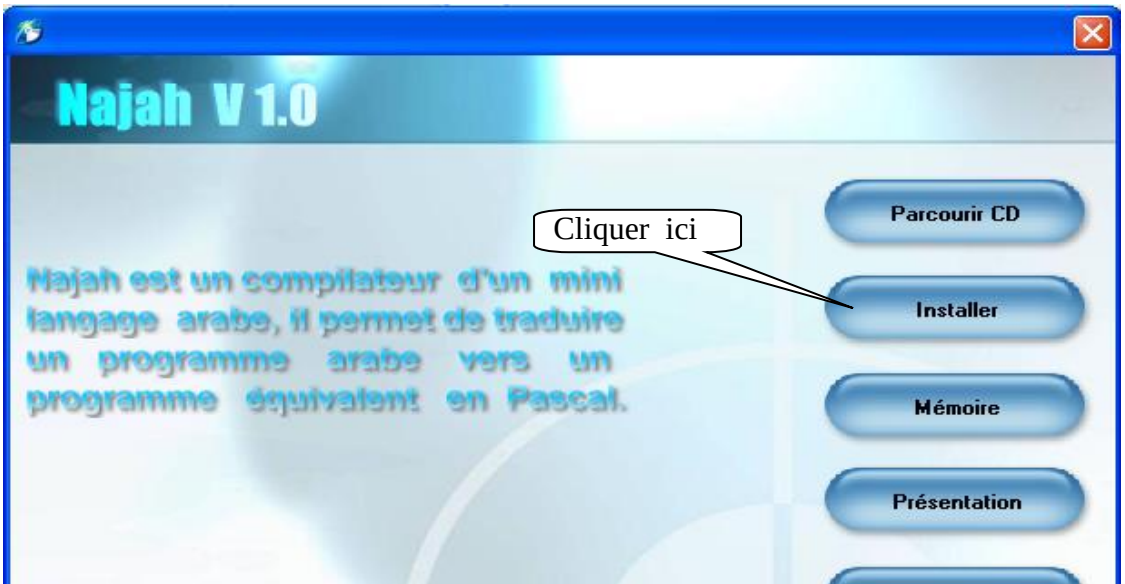


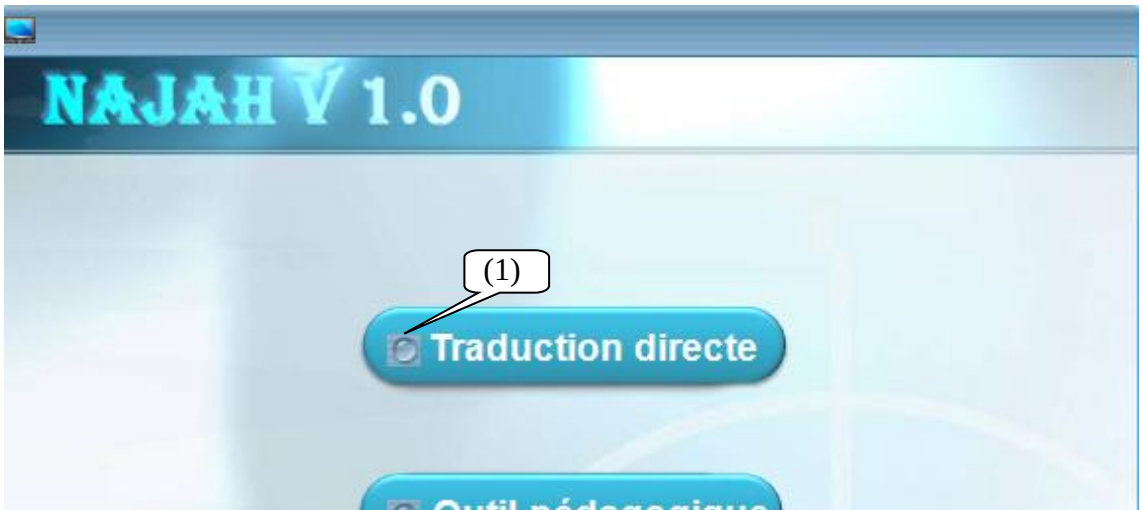
Figure 4.1 : la fenêtre d'installation du Najah

Après l'installation du Najah, son icône est ajoutée sur le bureau Windows et le menu démarrer.

- Lancer l'application en cliquant sur l'icône de l'application dans le bureau Windows.
- La fenêtre d'accueil de Najah s'affiche comme suit :

Figure 4.2: la fenêtre d'accueil du Najah

- En suite, la fenêtre principale s'affiche (voir la figure 4.3), elle offre le choix entre deux types de compilation, traduction directe « en arabe» comme il est illustré dans la figure 4.4, et le deuxième outil pédagogique « en français ».



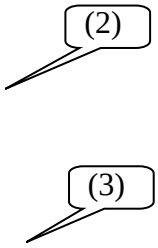
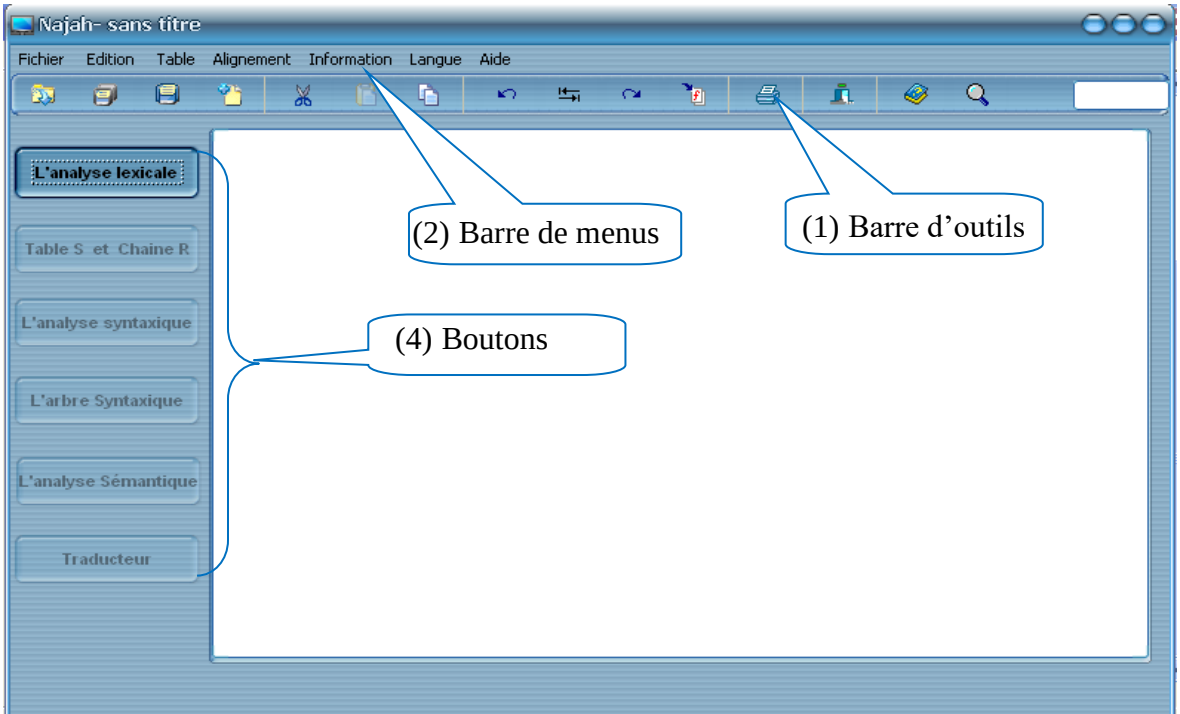


Figure 4.3 : la fenêtre principale de l'application

(1) : Pour permettre à l'utilisateur de suivre la compilation étape par étape, cliquer sur le bouton (1) pour afficher la fenêtre 4.4.

(2) : Pour réaliser une compilation d'un programme en une seule étape.

(3) : Quitter l'application.



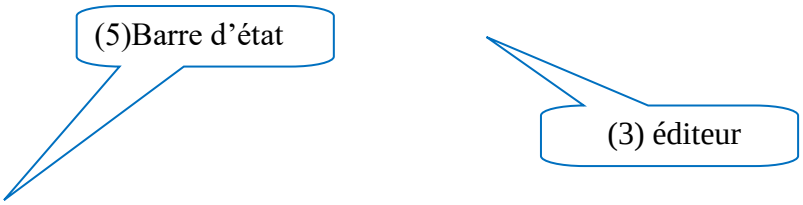


Figure 4.4 : la fenêtre du Najah « en français »

(1)La barre d'outils

Donne l'accès rapide aux fonctionnalités les plus utilisées telles que l'ouverture et sauvegarde des fichiers à l'aide des boites de dialogue, les commandes standard (Couper, Copier, Coller, Impression, Recherche, …).

(2)La barre de menus

L'interface de notre application contient des menus chaque menu permet de lancer un ensemble de commandes, concernant la gestion des fichiers (menu Fichier), les options de manipulation du texte (menu Edition) et un menu table affichant la structure des tables utilisées par le langage.

(3) L'éditeur

Est une zone de texte permettant l'utilisateur à d'écrire son programme.

(4) Les boutons d'actions

Représentent les étapes successives d'exécution (l'analyse lexicale, l'analyse syntaxique, l'analyse sémantique et la traduction) d'un programme écrit en mini langage « Najah » et leurs outils de conception (la table de symboles, la chaine de références et l'arbre syntaxique)

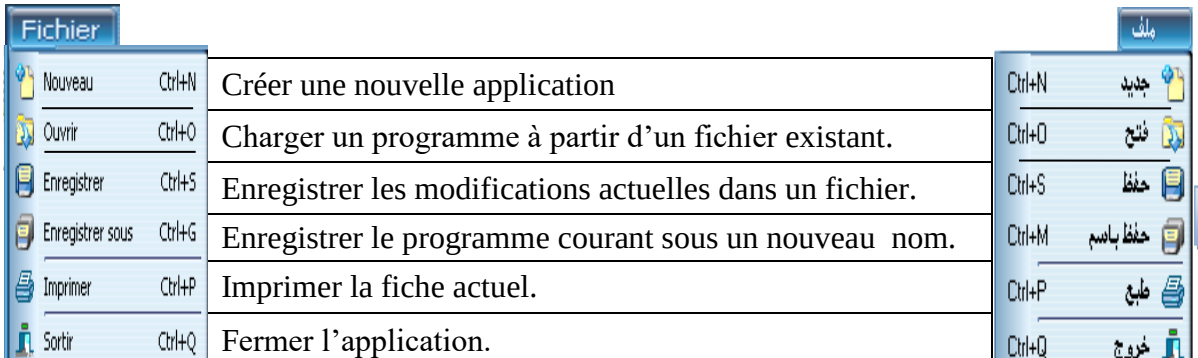
(5) Barre d'état

Affiche les différentes propriétés d'un programme tel que la position de curseur « colonne et ligne », l'état d'un programme « modifier ou non ».

4.6 La description détaillée de la barre de menus

a) Le menu Fichier

Le menu Fichier offre des commandes qui nous aident à manipuler des fichiers. Najah utilise une extension particulière (.naj) pour sauvegarder et ouvrir son code source.

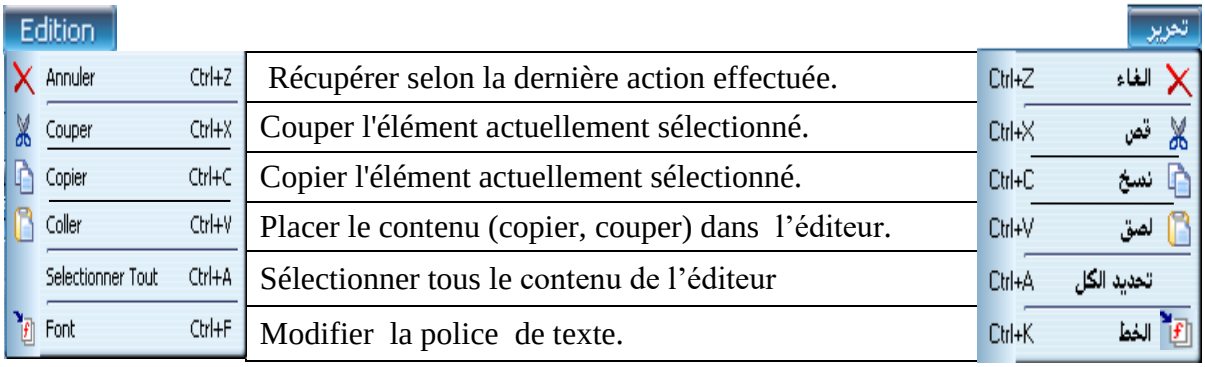


Créer une nouvelle application	
Charger un programme à partir d'un fichier existant.	
Enregistrer les modifications actuelles dans un fichier.	
Enregistrer le programme courant sous un nouveau nom.	
Imprimer la fiche actuel.	
Fermer l'application.	

Figure 4.5 : Le menu Fichier et son contenu

b) Le menu Edition

Le menu Edition contient plusieurs commandes (comme indiqué dans la Figure 4.6) pour manipuler le texte.



Annuler Ctrl+Z	Récupérer selon la dernière action effectuée.
Couper Ctrl+X	Couper l'élément actuellement sélectionné.
Copier Ctrl+C	Copier l'élément actuellement sélectionné.
Coller Ctrl+V	Placer le contenu (copier, couper) dans l'éditeur.
Selectionner Tout Ctrl+A	Sélectionner tous le contenu de l'éditeur
Font Ctrl+F	Modifier la police de texte.

Figure 4.6 : Le menu Edition et son contenu

c) Le menu Table

Le menu Table permet d'afficher les structures utilisées dans l'analyse lexicale et syntaxique, comme il est illustré dans la figure 4.7

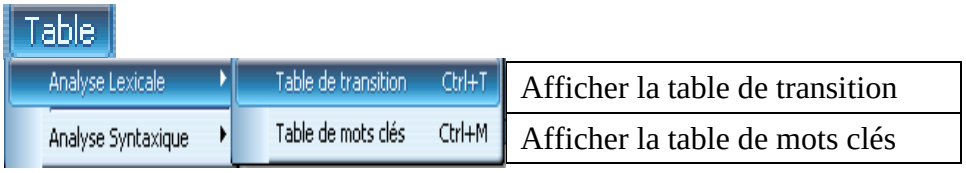


Table de transition Ctrl+T	Afficher la table de transition
Table de mots clés Ctrl+M	Afficher la table de mots clés

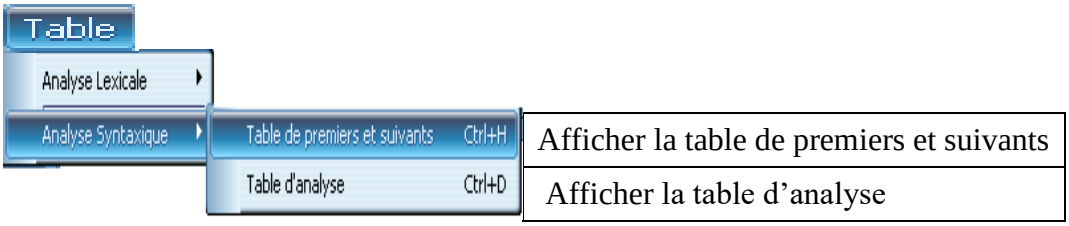


Table de premiers et suivants Ctrl+H	Afficher la table de premiers et suivants
Table d'analyse Ctrl+D	Afficher la table d'analyse

Figure 4.7 : Le menu Table et son contenu

d) Le menu Alignement

Le menu Alignement permet d'aligner un texte sélectionné vers (gauche, droite, centre).

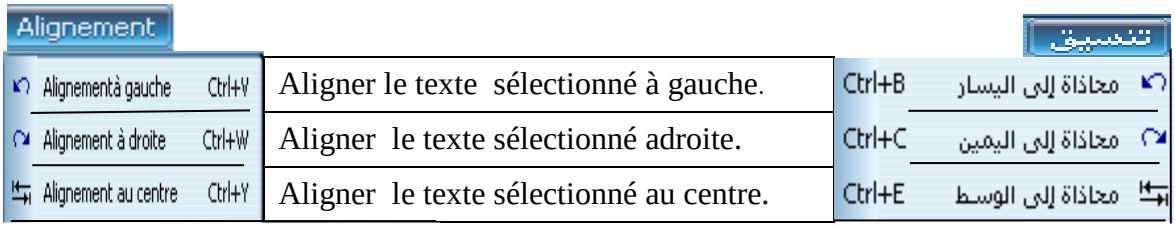


Figure 4.8: Le menu Alignement et son contenu

e) Le menu Information

Le menu Information contient les informations suivants : nom de programme, taille de programme, nombre de lignes, nombre des erreurs, la date et l'heure de la dernière d'enregistrement.

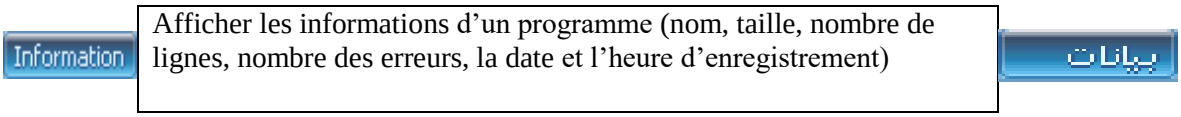


Figure 4.9 : Le menu Information et son contenu

f) Le menu Langue

Le menu Langue permet de changer la langue de l'éditeur vers (arabe, anglais, français).

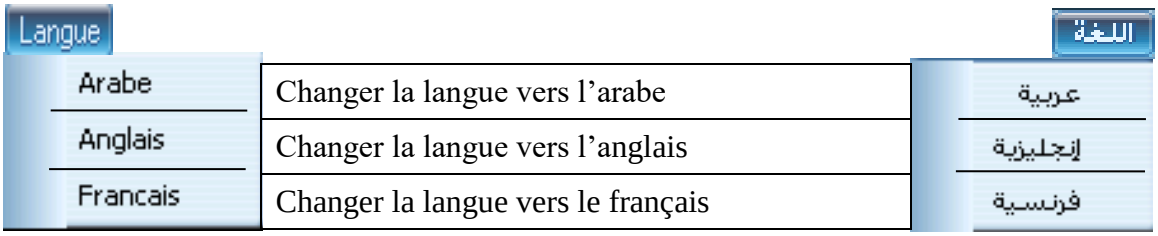


Figure 4.10 : Le menu Langue et son contenu

g) Le menu Aide

Le menu Aide permet d'afficher la forme d'aide associe au compilateur, ou les informations concernant l'application.

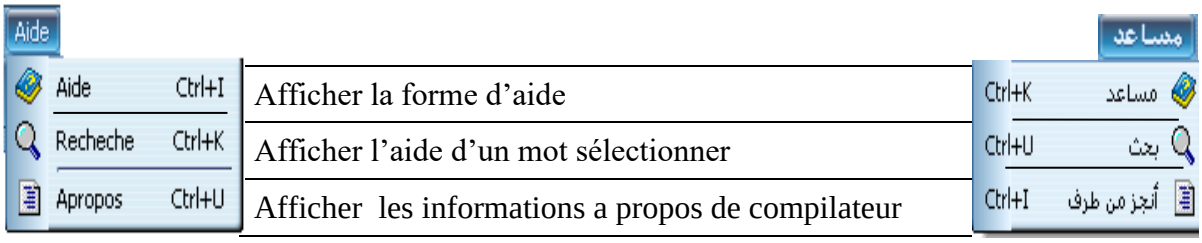


Figure 4.11 : Le menu Aide et son contenu

4.7 Exemple d'exécution

Cet exemple permet de calculer la valeur absolue d'un nombre entier. Mais il contient des erreurs :

```
برنامج حساب القيمة_المطلقة ؛
متغيرات
أ ، ب ، ن : حقيقي ..... (E2)
وظيفة حساب(1نabب: حقيقي ):حقيقي ؛.....(E1)
إبد .....(E3)

إذاكان (ن>=0) فإن
إبدأ حساب:=ن؛ إنتهى
وإلا
إبدأ حساب:=- ن؛ إنتهى؛
إنتهى؛
إبدأ
إقرأ_سطر ( ن )؛
ب:=مناداة_وظيفة حساب ( ن،ج ) ؛.....(E4 ; E5)
أكتب_سطر (ب)؛
إنتهى .
```

(E1) : Erreur lexical : **1نabcي** identificateur non accepte.

(E2) : Erreur syntaxique : l'absence d'un point virgule.

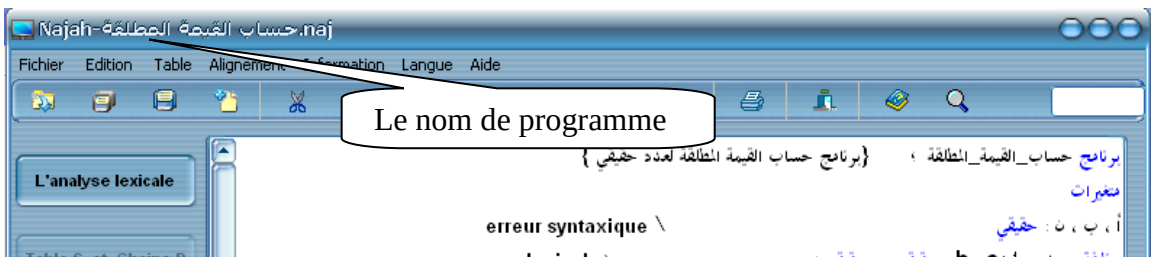
(E3) : Erreur syntaxique : mot réservé mal écrit.

(E4) : Erreur sémantique : le nombre des paramètres effectifs n'égale pas le nombre des paramètres formels de la fonction **حساب.**

(E5) : Erreur sémantique : Variable non déclaré.

On essaye de compiler ce programme à l'aide du Najah.

- Le programme est chargé dans l'éditeur, comme il est illustré dans la figure suivante :



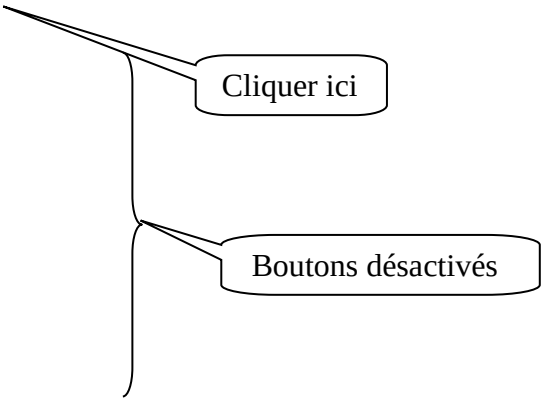


Figure 4.12: Exemple d'un programme qui calcul la valeur absolu d'un nombre réel

➢ On commence l'exécution par la première étape de la compilation, l'analyseur lexical détecte et signale l'erreur E1 : nature d'erreur, sélectionner l'unité lexicale qui contient l'erreur, numéro d'erreur, numéro de ligne d'erreur, possibilité de correction, comme il est illustré dans la figure 4.13.

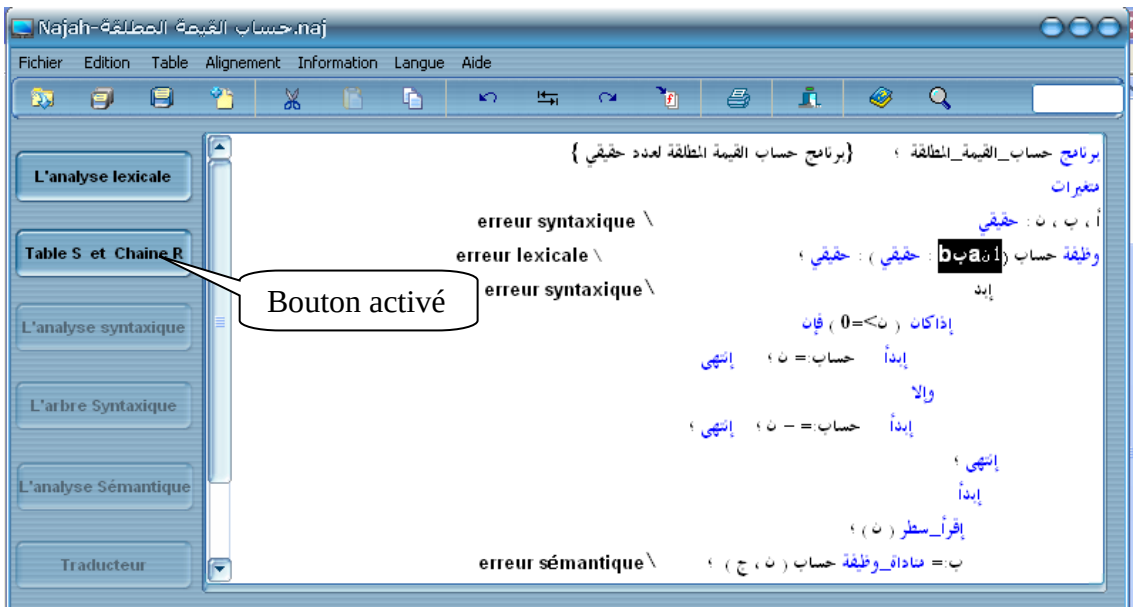
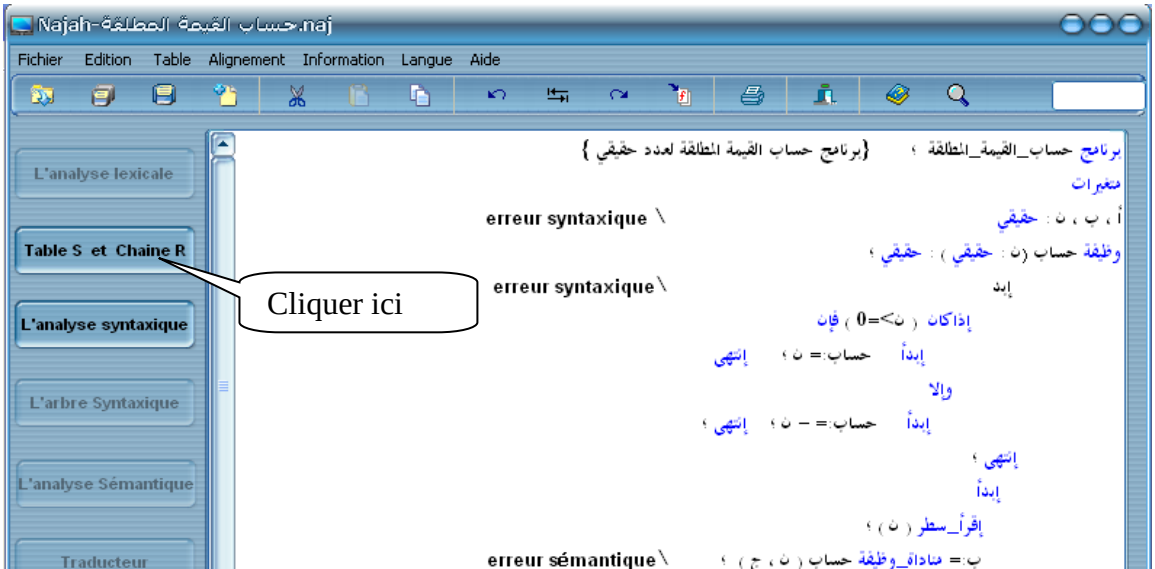


Figure 4.13 : résultat d'exécution de l'analyse lexicale (avec erreur)

➢ Après la correction d'erreur, on relance l'analyse lexicale, le résultat est affiché dans la figure4.14.



L'état de programme

Figure 4.14 : résultat d'exécution de l'analyse lexicale (sans erreur)

➢ L'exécution est terminé sans aucune erreur, on peut passer a l'étape suivante, le bouton' analyse syntaxique' devient actif et si on veut voir les structures de données internes telles que la table des symboles et la chaine de références on peut cliquer sur le bouton ' Table S et Chaine R ', comme est illustré dans la figure 4.15.

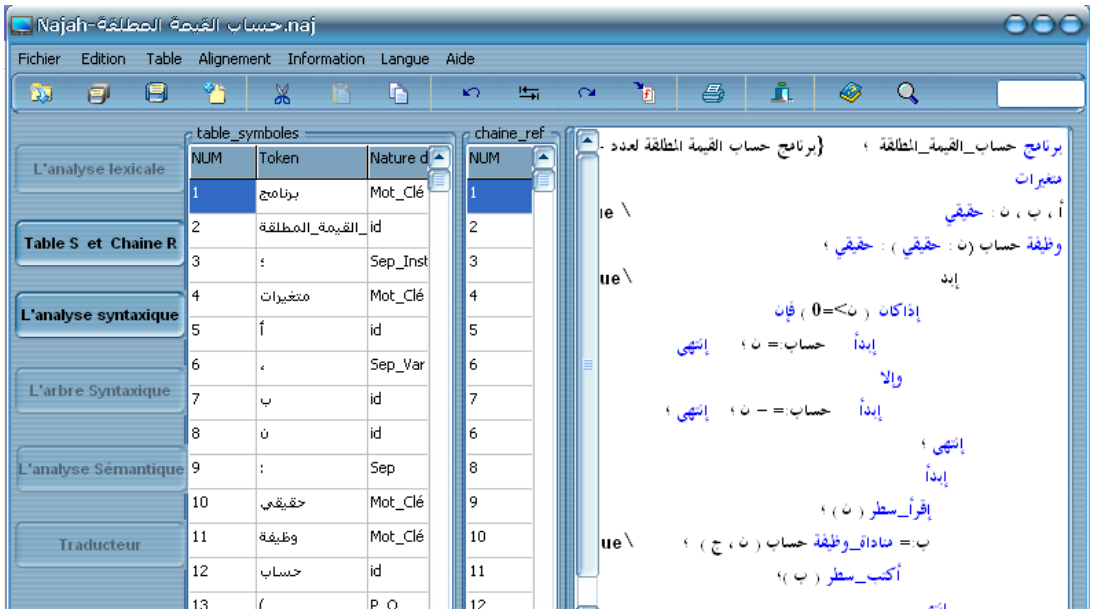


Figure 4.15: L'affichage de la table des symboles et la chaine de références

➢ Après le lancement de l'analyse syntaxique, deux erreurs sont détectées, E2 et E3 : nature d'erreur, numéro d'erreur, numéro de ligne d'erreur, sélectionner la ligne d'erreur.

L'analyseur syntaxique permet d'ajouter ';' pour corriger E2 et de

donner la possibilité de correction de E3, la figure 4.16 et la figure4.17 illustrent cette étape.

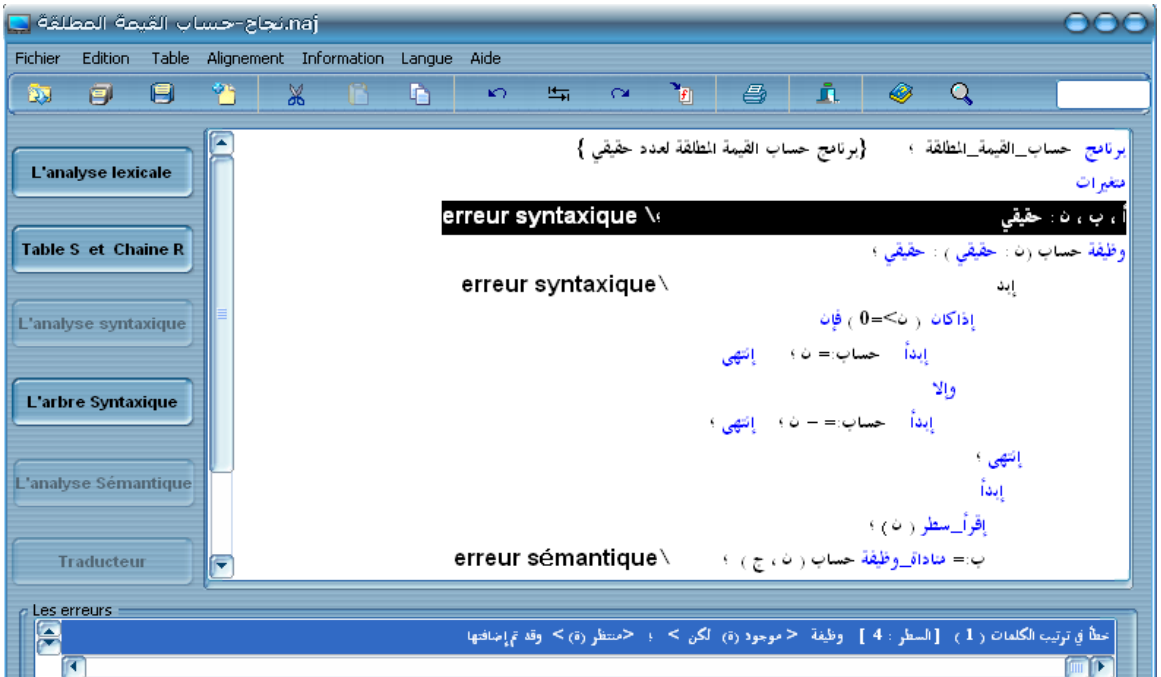


Figure 4.16: résultat d'exécution de l'analyse syntaxique (avec E2)

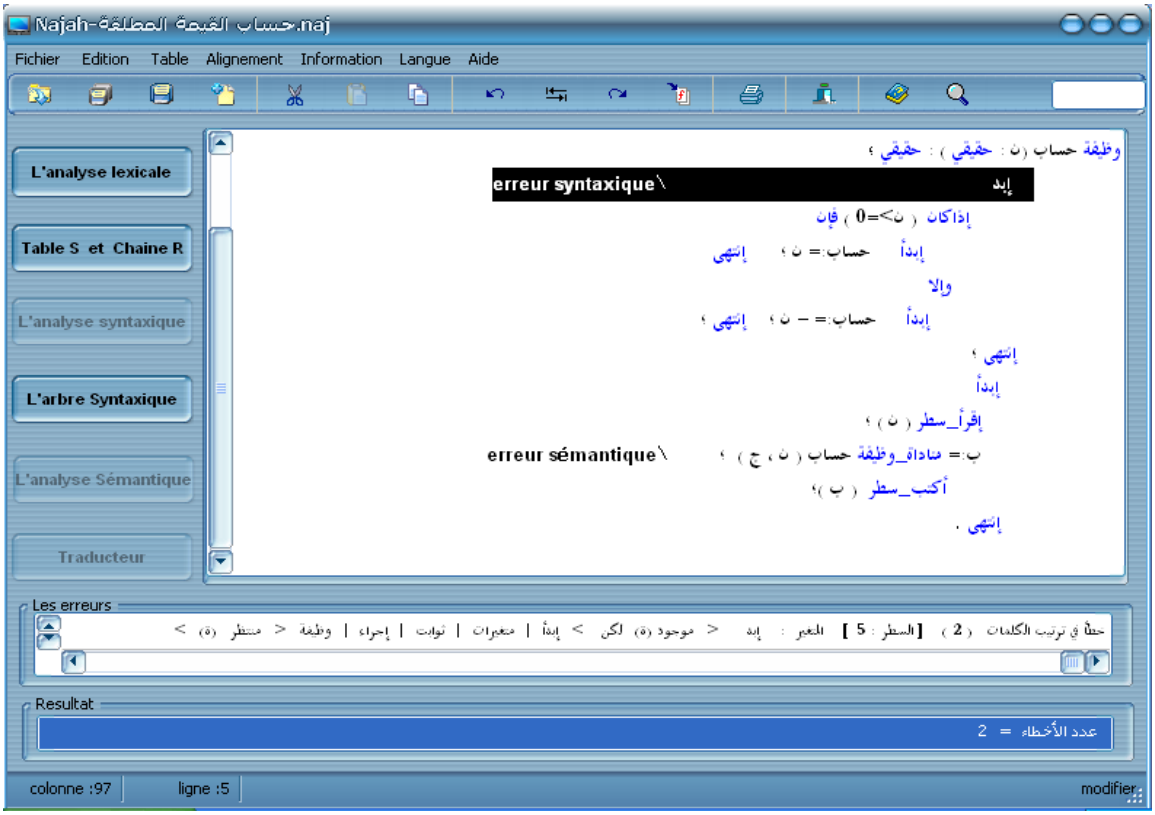


Figure 4.17: résultat d'exécution de l'analyse syntaxique (avec E3)

➢ Après la correction des erreurs, on relance l'analyse syntaxique, le résultat est affiché dans la figure suivante :

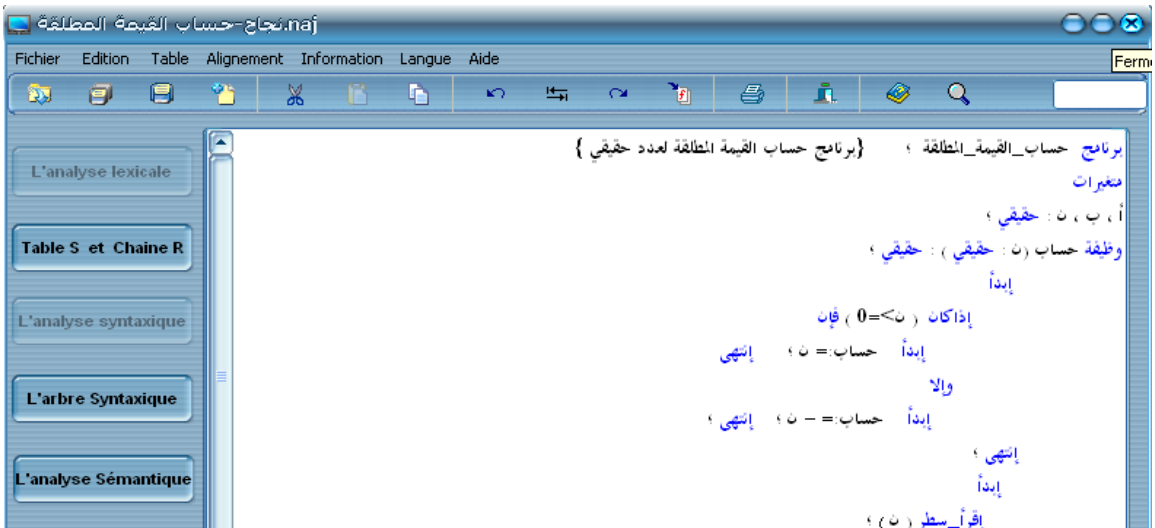
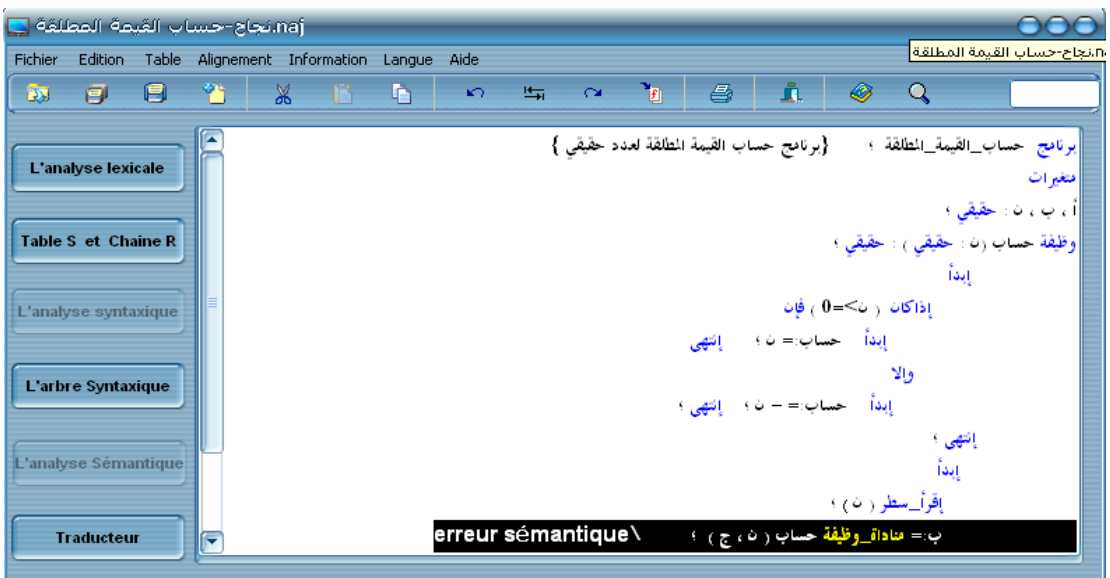


Figure 4.18 : résultat d'exécution de l'analyse syntaxique (sans erreur)

➢ L'exécution est terminé sans aucune erreur, on peut passer a l'étape suivante, le bouton' analyse sémantique' devient actif, on clique sur ce bouton.

L'analyseur sémantique détecte et signale les erreurs E4 et E5: nature d'erreur, numéro d'erreur, numéro de ligne d'erreur, sélection du ligne d'erreur, possibilité de correction comme il est illustré dans la figure4.19 et la figure4.20.



خطأ في معنى الكلمات (1) [السطر:13] <لا يوجد توافق في عدد المتغيرات بين مناداة_وظيفة و الوظيفة>

Figure 4.19 : résultat d'exécution de l'analyse sémantique (avec E4)

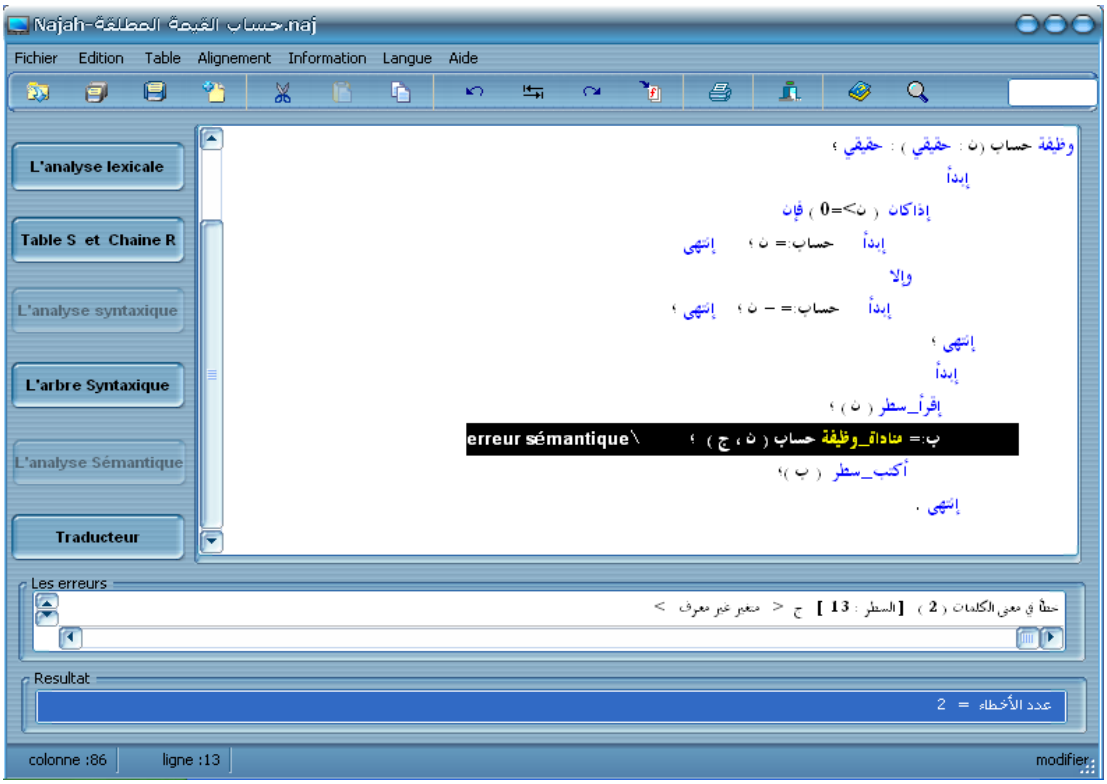


Figure 4.20 : résultat d'exécution de l'analyse sémantique (avec E5)

- Après la correction des erreurs, on relance l'exécution, le résultat est affiché dans la figure 4.21.

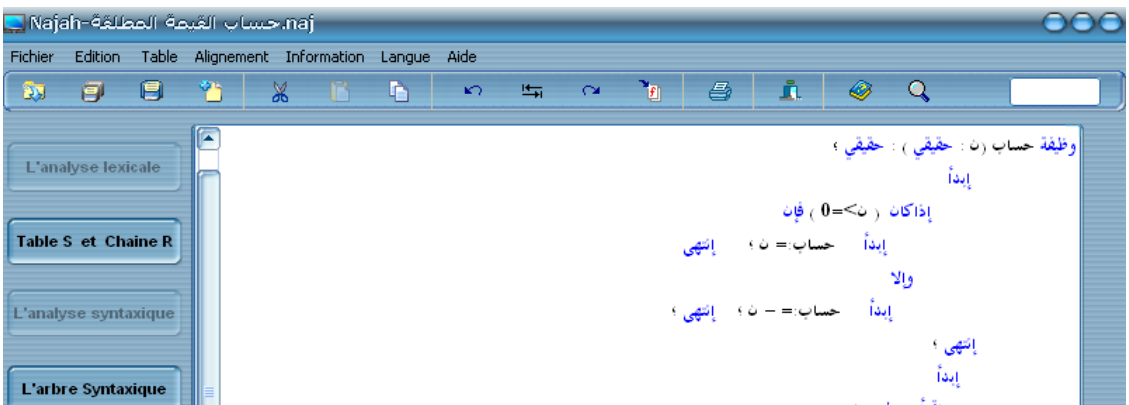


Figure 4.21: résultat d'exécution de l'analyse sémantique (sans erreur)

➢ A la fin de toutes ces étapes, on remarque que le bouton' traduction 'devient actif, cliquer sur ce bouton la fenêtre de traduction s'affiche dans la figure 4.22.

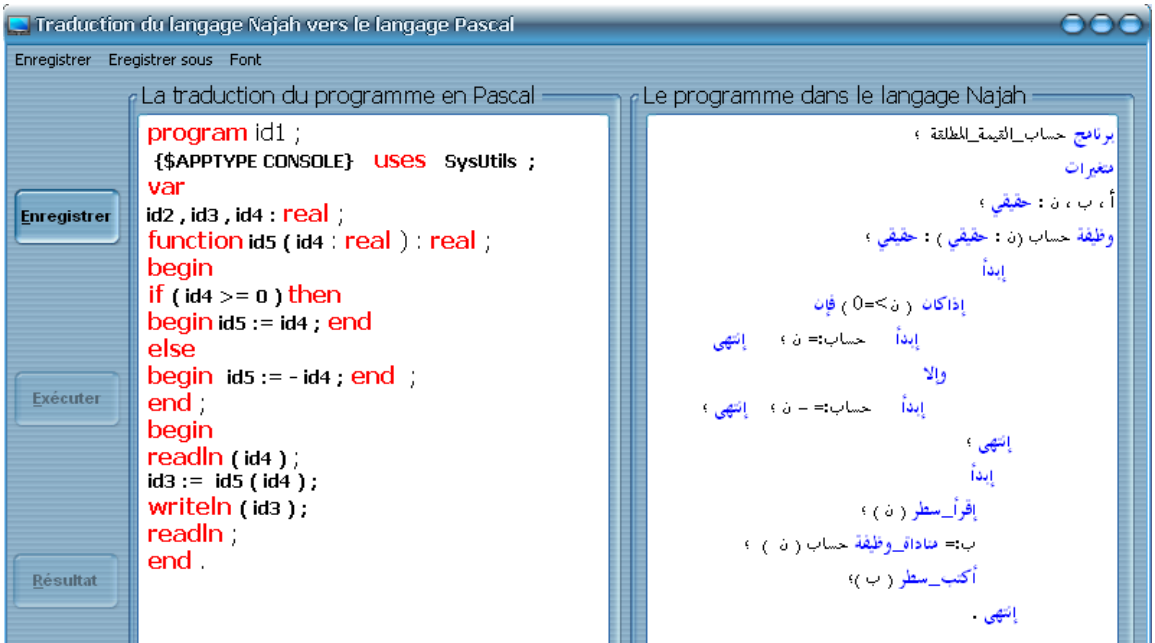


Figure 4.22 : La traduction du programme en langage Najah vers le langage Pascal

Notre but était de générer un code source à partir de notre langage. On a dépassé cette étape à une étape plus avancée qui consiste à exécuter le code source résultant, on est sûr que le programme résultant ne contient aucune erreur parce que toutes les erreurs ont été traitées par notre compilateur, nous suivons les étapes suivantes pour exécuter le code traduit:

- Cliquer sur le bouton 'Sauvegarder' pour sauvegarder le code traduit,
- Appuyer sur le bouton 'Exécuter' pour exécuter le code enregistré,
- Appuyer sur le bouton 'Résultat' pour appeler l'exécutable qui est enregistré dans le même répertoire que le code traduit.
- En fin, on fait des tests, on entre la valeur – 21 par exemple, et on remarque que le résultat de la valeur absolue de -21 est égale à 21. comme il est illustré dans la figure 4.23.

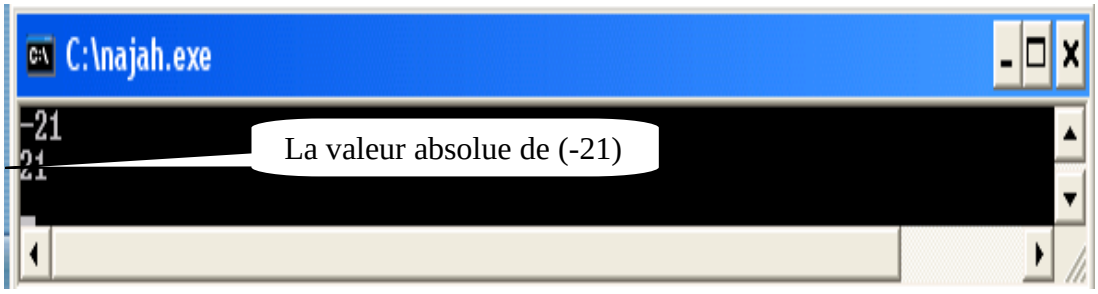


Figure 4.23: L'exécutable du programme

Si on veut exécuter cet exemple sans passé par les différentes étapes de compilation, on peut l'exécuter directement :

- Cliquer sur le bouton de deuxième choix de la figure 4.13 la fenêtre suivante s'affiche.
- Charger le programme.

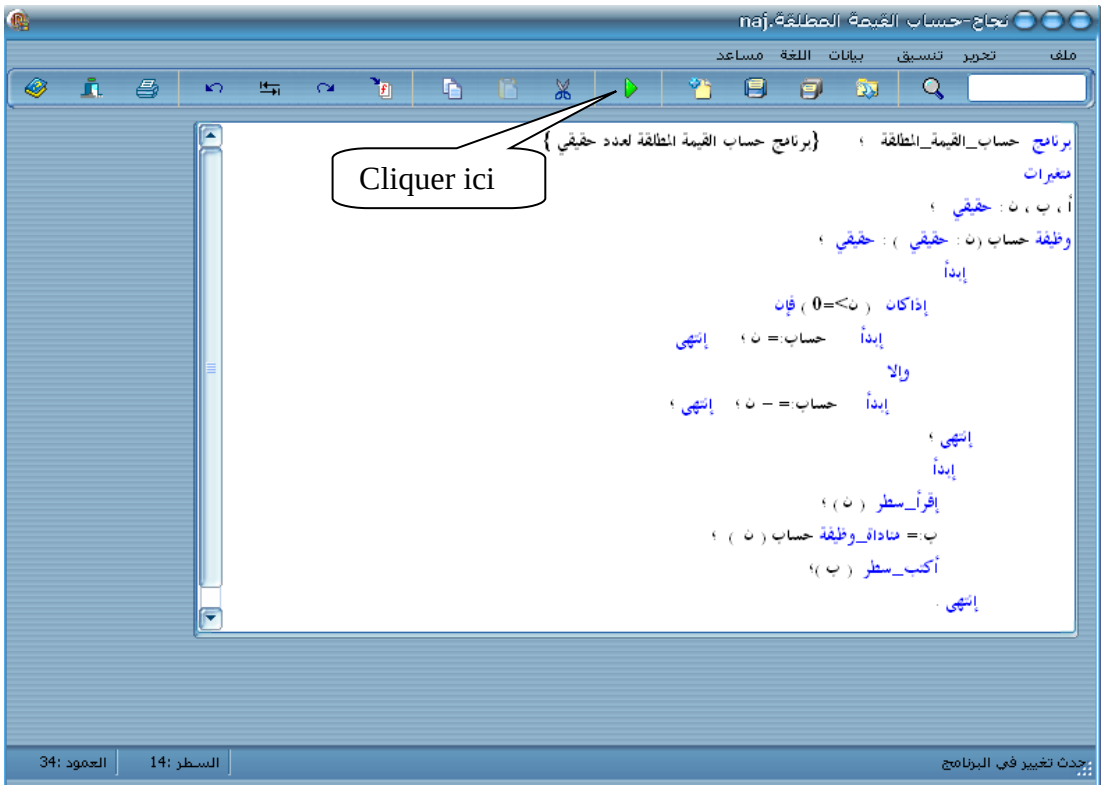


Figure 4.24: la fenêtre d'exécution directe du programme

 Regroupe le fonctionnement de tous les boutons de la figure 4.4.

Après la clique sur le bouton  on obtient le résultat directement. Comme il est illustré dans la figure 4.23.

4.8 Aide du logiciel

L'application du Najah est enrichie par une grande partie d'aide. Elle contient deux fichiers d'aide, le premier pour la partie française qui présente le manuel d'utilisation de notre application. Comme il est illustré dans la figure 4.25, et le deuxième pour la partie arabe qui présente le guide d'utilisation de notre langage (voir la figure 4.26).

Figure 4.25: Fenêtre d'aide en français

Figure 4.26: Fenêtre d'aide en arabe

➢ Si vous voulez voir l'aide d'un mot réservé, taper le mot dans l'édit puis cliquer sur le bouton 'chercher' .

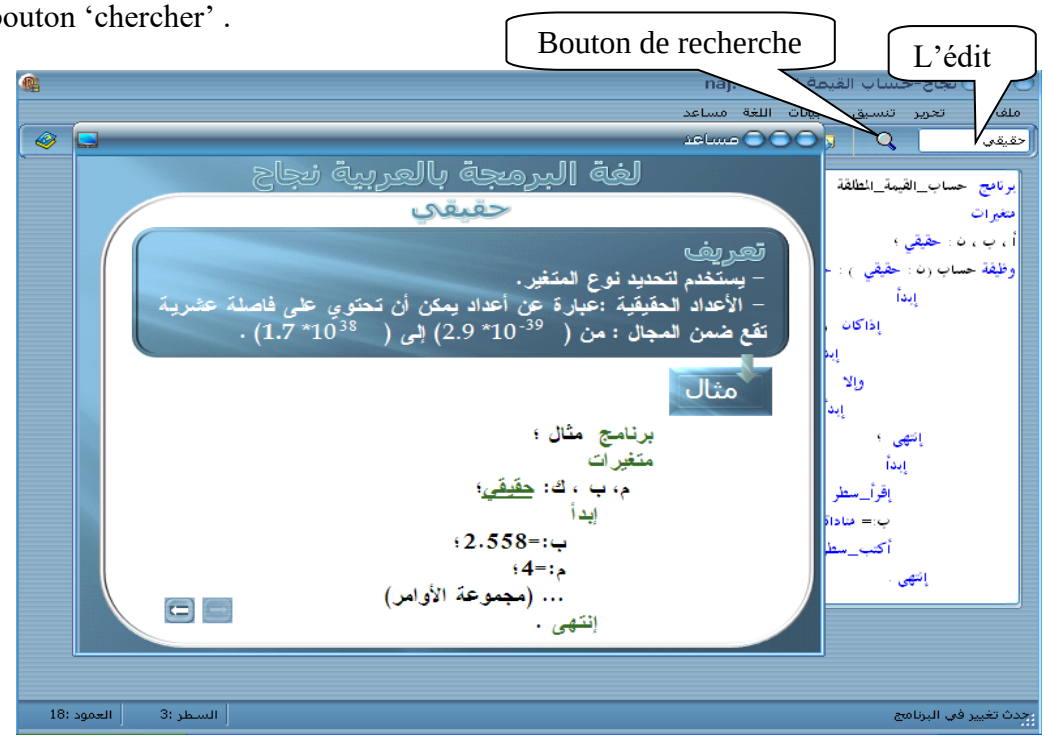


Figure 4.27: Fenêtre d'aide d'un mot

➢ Si vous voulez voir les informations concernant le programme, cliquer sur le menu 'بيانات' .

Figure 4.28: Fenêtre d'information

CONCLUSION

'La marche des 1000 milles commence par un pas'

Le but principal de notre travail était de développer un compilateur d'un mini langage de programmation arabe, pour cela nous avons commencés par étudier les langages de programmation, leur historique et leur développement.

Ensuite, nous avons étudiés les principes de bases de la compilation : l'analyse lexicale, l'analyse syntaxique, l'analyse sémantique et le processus de traduction avec les outils fondamentaux utilisés pour effectués ces analyses : fondement de base de la théorie des langages (grammaire, automate, …), méthodes algorithmique des différentes étapes de compilation.

Dans ce cadre, nous avons développé le compilateur 'Najah V 1.0 'qui traduit un programme écrit dans un mini langage de programmation arabe vers un programme écrit dans le langage de programmation Pascal.

Nous avons décomposé notre application en deux parties : la première « étape par étape » avec une interface en français à pour but de présenter les différentes étapes et outils de la compilation pour faciliter l'enseignement du module de compilation dans le domaine de l'informatique.

La deuxième partie « toutes les étapes » avec une interface en arabe à pour but de faciliter la tâche aux élèves et aux lycéens à comprendre les principes et les notions d'algorithmique et d'apprendre la programmation.

Ce projet nous a permis :

- ✓ De bien comprendre et appliquer les différents techniques et méthodes d'analyse, de la compilation et de la programmation.
- ✓ De bien maitriser le logiciel de programmation Delphi 2007.
- ✓ D'avoir une bonne expérience dans le domaine de recherche.

Pour cela, il est intéressant d'améliorer Najah V 1.0 pour obtenir une nouvelle version contenant les nouvelles fonctionnalités suivantes :

- ✓ L'ajout des autres fonctions qui sont préfinis par Pascal (par exemple : case, Rondom,…)
- ✓ L'ajout des types comme : boolean, word…
- ✓ L'ajout des structures de données, par exemple : les pointeurs.
- ✓ Traitement des fichiers.
- ✓ La traduction vers les autres langages tel que: Java, C++,…
- ✓ L'adaptation du langage pour d'autres plateformes tel que : Linux, Unix, …

ANNEXES

ANNEXE 1

LES STRUCTURES DE DONNÉES

ET LES ALGORITHMES UTILISÉS

Cette annexe représente les structures de données utilisées et les différentes procédures appliquées.

A1.1 Les structures de données dans notre code

✓ la table de mots clés

On regroupe tous les mots clés dans un vecteur qui représente la table de mots clés.

✓ la table de transition et la table de premiers et suivants

Sont représentés par des matrices de deux dimensions.

✓ la table d'analyse

L'analyseur syntaxique est dirigé par la table d'analyse, on utilise une matrice de deux dimensions pour représenter cette table et une pile pour déterminer si le mot à analyser appartient au langage.

✓ la table des symboles

Une fonction essentielle d'un compilateur est d'enregistrer les unités lexicales utilisées dans le programme source et de collecter de l'information sur divers attributs de chaque unité lexicale.ces attributs fournissent de l'information concernant par exemple, l'emplacement mémoire assigné a un identificateur, sa nature, son type. Pour cella on utilise une liste pour représenter la table des symboles contenant les champs suivants :

Numéro	Token	Nature de token	Type de token	Adresse suivant

✓ la chaine de références

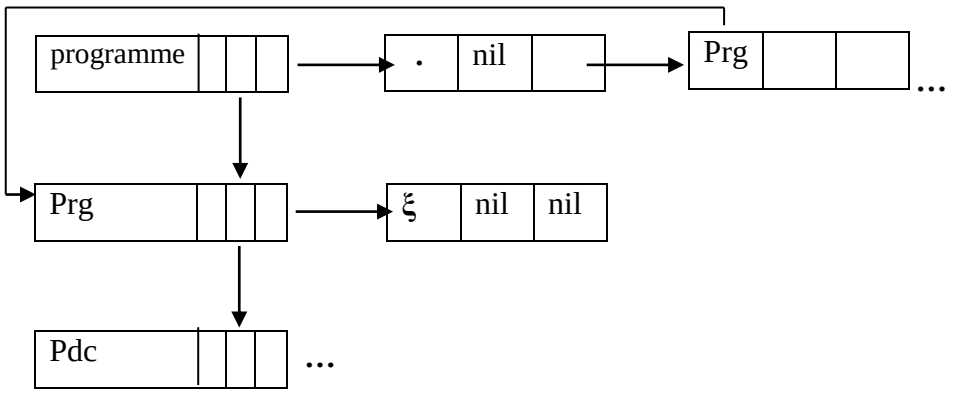
A la fin da la première partie de compilation, l'analyseur lexical génère une suite des unités lexicales qu'elles sont regroupées dans une chaine de références, la chaîne est remplir au fur et à mesure du remplissage de la table de symboles. On utilise une liste pour représenter cette chaine.

Numéro	Adresse de Suivant

✓ L'arbre syntaxique

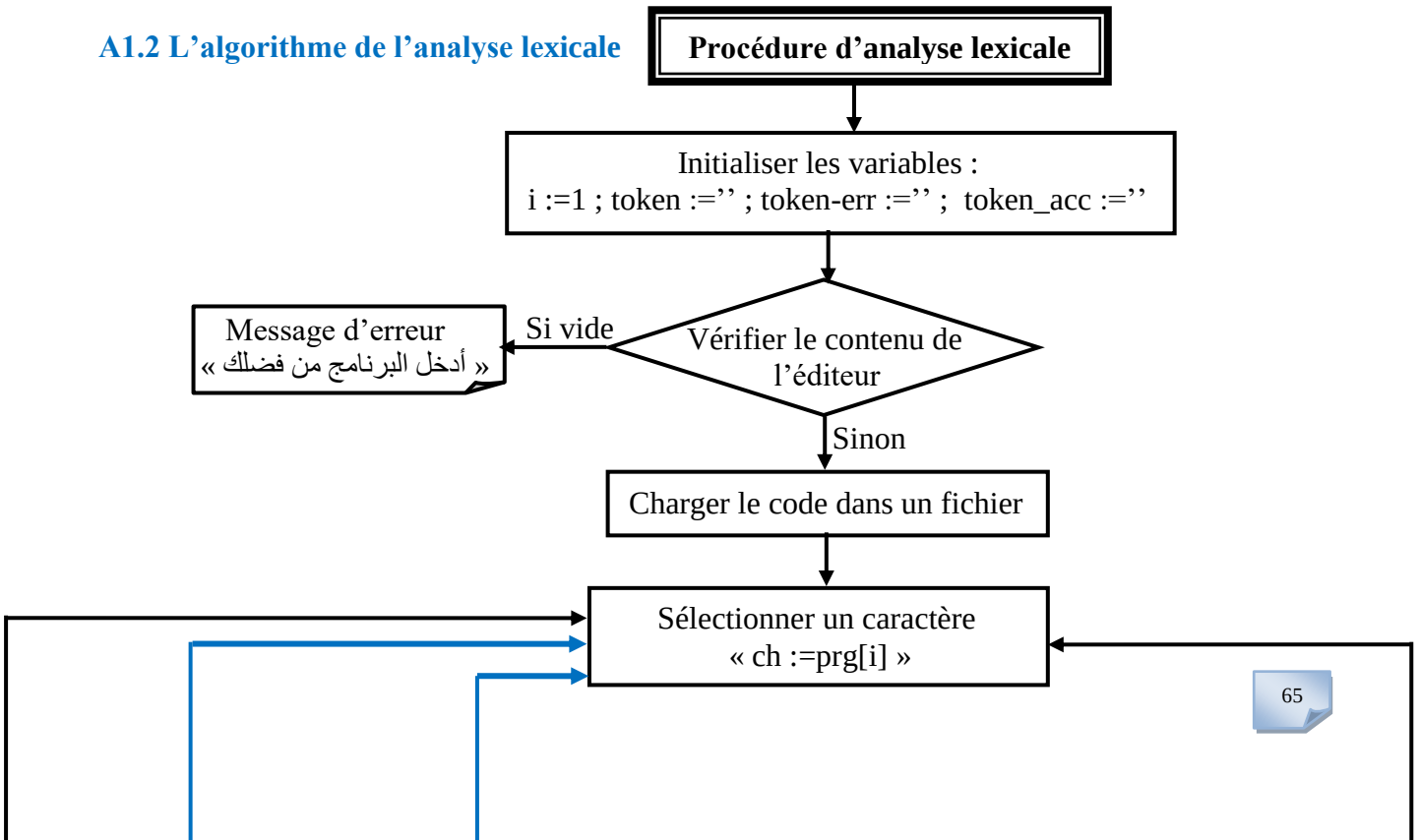
Est représenté par une liste linéaire de sommets « contient les non terminaux de la grammaire » et chaque sommet en lui associe a une liste d'adjacence qui représente la partie droite de la règle de production.

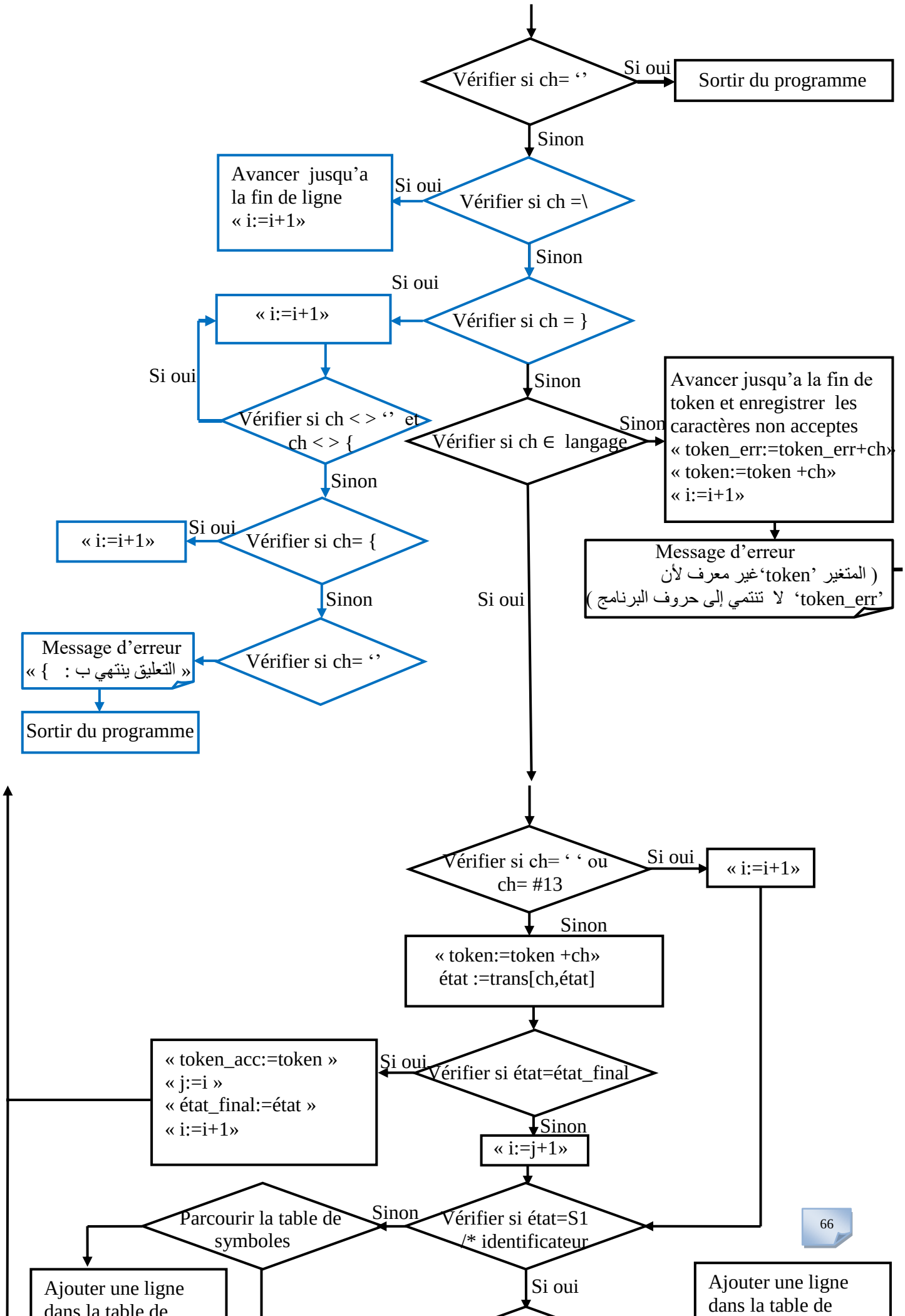
Exemple



…

A1.2 L'algorithme de l'analyse lexicale



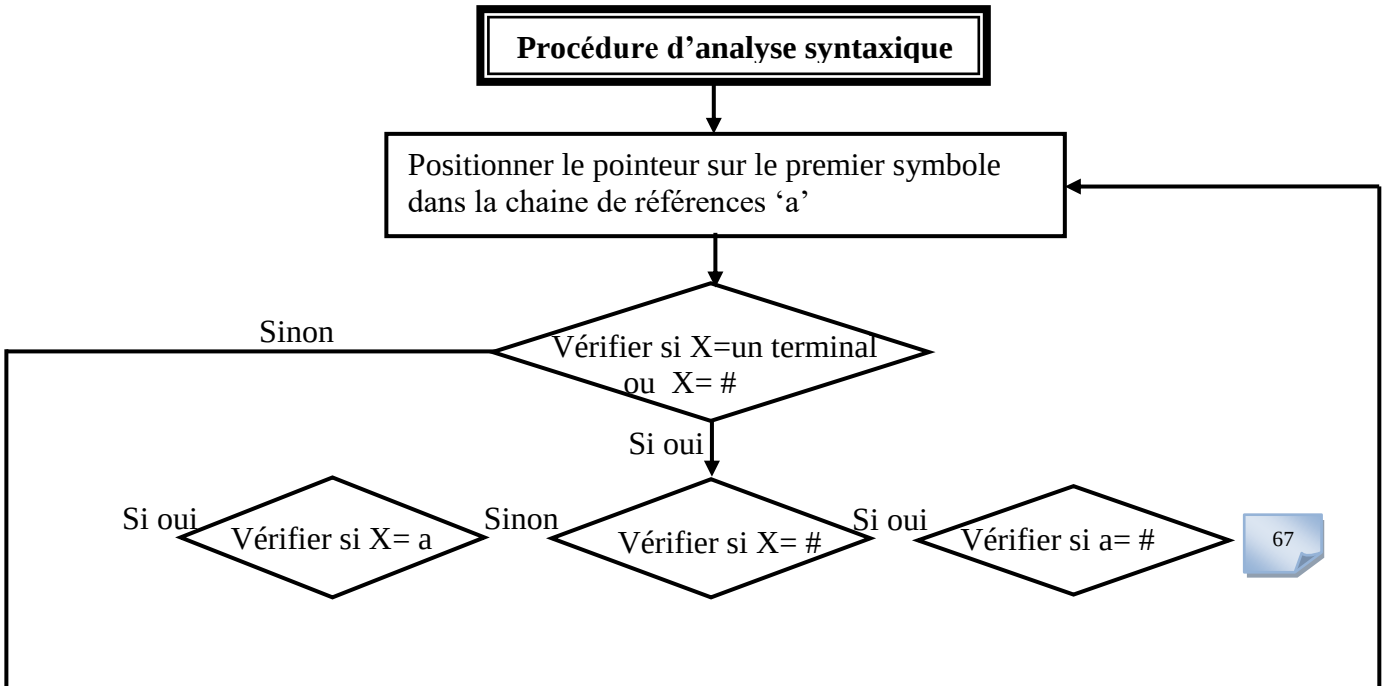

Vérifier si ch= ‘’
Si oui
Sortir du programme
Sinon
Avancer jusqu'a la fin de ligne « i:=i+1»
Si oui
Vérifier si ch =\
Sinon
Si oui
« i:=i+1»
Vérifier si ch = }
Si oui
Sinon
Avancer jusqu'a la fin de token et enregistrer les caractères non acceptes « token_err:=token_err+ch» « token:=token +ch» « i:=i+1»
Vérifier si ch < > ‘’ et ch < > {
Vérifier si ch ∈ langage
Sinon
Message d'erreur (المتغير ’token‘غير معرف لأن ’token_err‘ لا تنتمي إلى حروف البرنامج)
Sinon
« i:=i+1»
Si oui
Vérifier si ch= {
Si oui
Sinon
Message d'erreur « { : التعليق ينتهي ب »
Vérifier si ch= ‘’
Sortir du programme
Vérifier si ch= ‘ ‘ ou ch= #13
Si oui
« i:=i+1»
Sinon
« token:=token +ch» état :=trans[ch,état]
« token_acc:=token » « j:=i » « état_final:=état » « i:=i+1»
Si oui
Vérifier si état=état_final
Sinon
« i:=j+1»
Parcourir la table de symboles
Sinon
Vérifier si état=S1 /* identificateur
Si oui
Ajouter une ligne dans la table de
Ajouter une ligne dans la table de

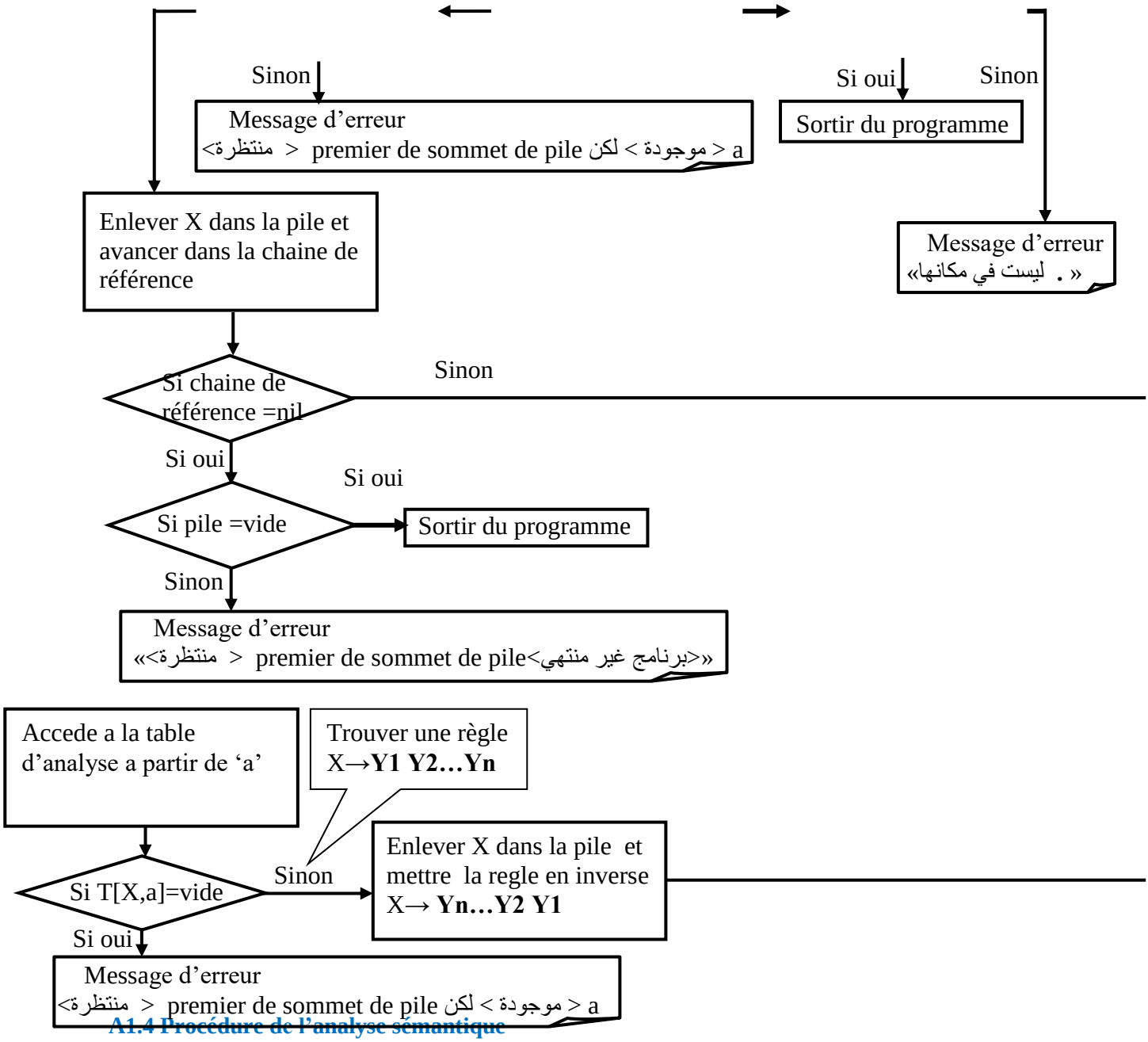
Parcourir la table de
Mots clés

Remarque : —— Traitement de commentaire

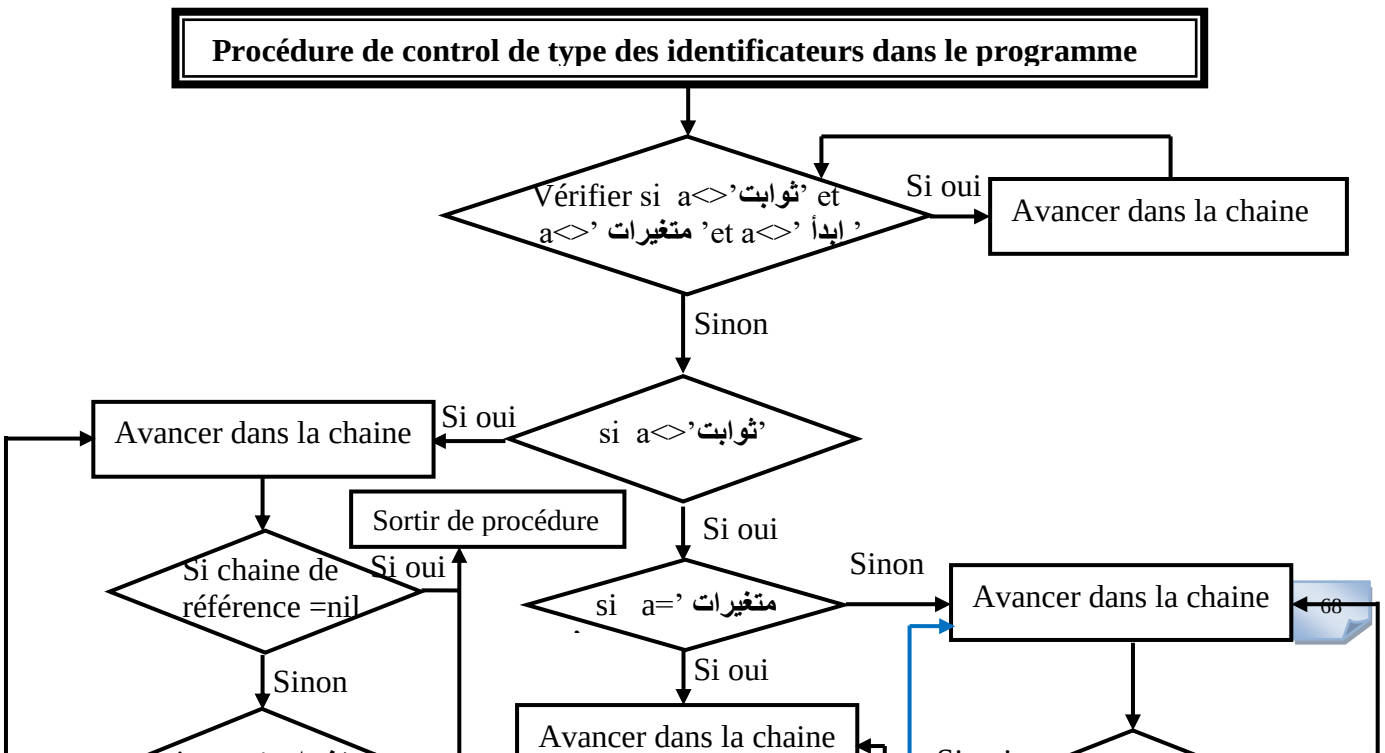
A1.3 L'algorithme de l'analyse syntaxique

Soit 'X' le symbole en sommet de pile et 'a' le premier symbole dans la chaine de référence.

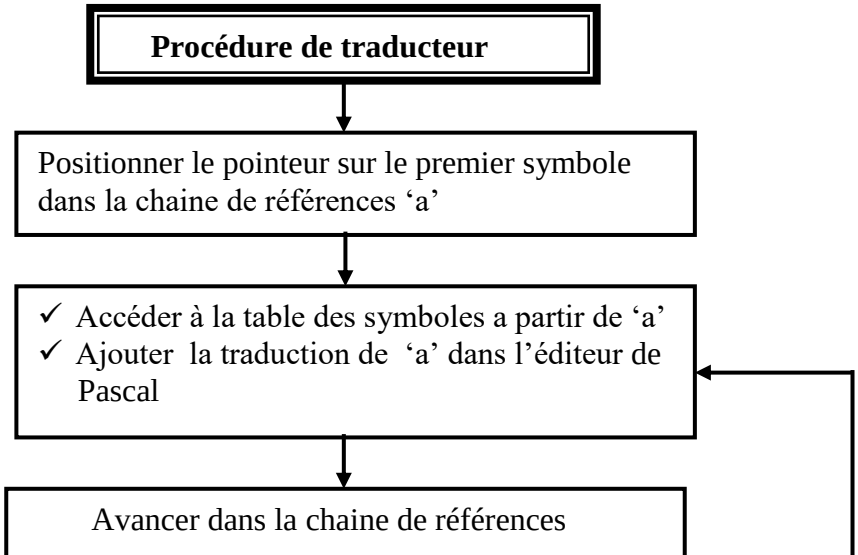


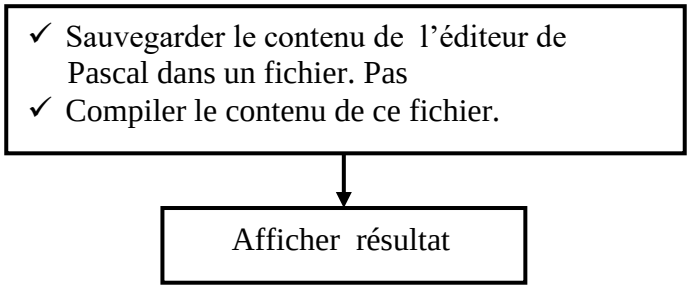


A1.4 Procédure de l'analyse sémantique



A1.5 Procédure de traducteur





ANNEXE 2

STRUCTURE GÉNÉRALE

DE LANGAGE NAJAH

Afin de mieux comprendre et bien maîtriser notre mini langage, on vous propose la structure générale de notre mini langage de programmation

A2.1 Structure générale de notre langage

Un programme NAJAH est composé d'un entête, des déclarations et des instructions (délimitées par **إبدأ** et **.إنتهى**).

EXEMPLE 1

```
                                    {entête}        برنامج دائرة ؛
                                    {Déclarations}  متغيرات
                                                    القطر ، المحيط : حقيقي ؛
                                                    إبدأ
{Instruction}                                       إقرأ_سطر(القطر) ؛
{Instruction}                                       المحيط :=  3.141592* القطر؛
{Instruction}                                       أكتب_سطر(القطر ، المحيط)؛
                                                    إنتهى .
```

A2.1.1 Les constantes

Nous avons déjà utilisés des variables (voir l'exemple précédent). Mais on peut également utiliser des constantes, qui sont des valeurs qui restent fixes tout au long du programme et pour chaque programme. On déclare les constantes avant de déclarer les variables.

EXEMPLE 2

```
ثوابت
قيمة_أ = 5؛
```

On peut également avoir des constantes de type "chaîne de caractères".

EXEMPLE 3

```
برنامج رسالة ؛
ثوابت
    رسالة_1 ="صباح الخير" ؛
إبدأ
    أكتب_سطر(رسالة_1)؛
إنتهى .
```

A2.1.2 Les types de variables et opérateurs associes

- ✓ **صحيح (entier)**

متغيرات **{Déclarations}**

أ ، ب ،ج : صحيح؛

Opérations sur entier : + − * /. Elles sont toutes à résultat entier, et nécessitent deux arguments entiers.

✓ **حقيقي (reel)**

متغيرات **{Déclarations}**

أ ، ب ،ج : حقيقي؛

Opérations : + − * /

Quand une opération comprend un argument réel et un entier, le résultat est réel.

✓ **حرف (caractère)**

متغيرات **{Déclarations}**

أ ، ب ،ج : حرف؛

Opérations : +

Ces variables contiennent UN caractère.

✓ **سلسلة (chaine de caractères)**

متغيرات **{Déclarations}**

أ ، ب ،ج : سلسلة؛

Opérations : +

Ces variables contiennent plusieurs caractères.

A2.1.3 Instruction d'affectation

On appelle affectation la mise d'une valeur dans une variable. Celle−ci peut être sous forme directe **(ب =:ج)** ou sous forme d'un calcul **(ج * ب =: ب)** « voir l'exemple 1 ».

A2.1.4 Instruction

On appelle instruction soit :

- ✓ une affectation.
- ✓ un appel à une procédure.
- ✓ une structure de contrôle.

On appelle "instruction composée" le regroupement de plusieurs instructions sous la forme:

إبدأ instruction1 instructionN...instruction2 إنتهى .

A2.1.5 Structure de contrôle

✓ **Boucle : مادام . . قم**

Structure: instructions قم expression booléenne مادام

Elle permet de répéter l'instruction tant que l'expression (ou la variable) booléenne est vraie.

EXEMPLE 4

```
برنامج حساب_أس ؛
متغيرات
أ ، ب ، ج ، د : صحيح ؛
إبدأ
إقرأ_سطر( أ ، ب ) ؛
د :=1؛
ج :=1؛
مادام ( ج <= ب ) قم
إبدأ
د:= د * أ ؛
ج:= ج+1؛
إنتهى ؛
أكتب_سطر( د)؛
إنتهى .
```

✓ **Boucle : أعد .. حتى**

Structure :

أعد

instruction1;

instruction2;

...etc...

instructionN

conditions حتى

Les N instructions sont répétés jusqu'à ce que la condition soit vérifiée. Même si la condition est vraie dès le début, elles sont au moins exécutées une fois

EXEMPLE 5

```
برنامج حساب ؛
متغيرات
أ ، ب : صحيح ؛
إبدأ
أكتب_سطر("أدخل قيمة أ أقل من 100")؛
إقرأ_سطر( أ ، ب ) ؛
أعد
أ:= أ +1 ؛
ب:= ب + أ ؛
حتى ( أ = 100) ؛
أكتب_سطر( ب)؛
إنتهى .
```

✓ **Boucle : لأجل . . قم**

Structure : instructions **قم** valeur_fin **إلى** valeur_début =: variable **لأجل**

La variable(non réelle) prend la valeur_début, et l'instruction est exécutée. Puis elle est incrémentée et ce jusqu'à valeur_fin.

EXEMPLE 6

```
برنامج حساب_عاملي ؛
متغيرات
أ ، ب ، ن : صحيح ؛
إبدأ
إقرأ_سطر( ن ) ؛
إذاكان ( ن<0 ) فإن
إبدأ أكتب_سطر("لا توجد نتيجة " )؛ إنتهى
وإلا إبدأ
ب :=1؛
لأجل أ :=1 حتى ن قم
إبدأ
ب:= ب * أ ؛
إنتهى ؛
أكتب_سطر("ن عاملي يساوي : " ، ب )؛ إنتهى ؛
إنتهى .
```

A2.1.6 Instruction conditionnelle

Instruction simple

structure : instructions فإن conditions إذاكان

Instruction imbriqué

structure : instructions وإلا instructions فإن conditions إذاكان

EXEMPLE 7

```
برنامج حساب_القيمة_المطلقة ؛
متغيرات
أ ، ب ، ن : صحيح ؛
إبدأ
إقرأ_سطر( ن ) ؛
إذاكان ( ن >=0 ) فإن
إبدأ
ب:= ن؛
إنتهى ؛
وإلا
إبدأ
ب:= - ن؛
إنتهى ؛
أكتب_سطر("القيمة المطلقة = " ، ب )؛
إنتهى .
```

A2.1.7 Les tableaux

On a souvent besoin de regrouper dans une seule variable (et donc un seul nom) plusieurs variables .On utilise pour cela les tableaux. La manière la plus simple des les définir est :

type من [Nombre2. . Nombre1] جدول : nom_tableau متغيرات

EXEMPLE 8

```
برنامج ترتيب ؛
متغيرات
أ :جدول[1..100] من حقيقي ؛
ب ، ج ، ن : صحيح ؛
م : حقيقي ؛
إبدأ
إقرأ_سطر( ن ) ؛
لأجل ب :=1 حتى ن قم
إبدأ
إقرأ ( أ [ ب ] ) ؛
إنتهى ؛
لأجل ب :=1 حتى ن -1 قم
إبدأ
لأجل ج := ب+1 حتى ن قم
إبدأ
إذاكان ( أ [ ب ] >أ [ ج ]) فإن
إبدأ
م:= أ [ ب ] ؛
أ [ ب ] :=أ [ ج ] ؛
أ [ ج ]:= م ؛
إنتهى ؛
إنتهى ؛
إنتهى ؛
لأجل ب := 1 حتى ن قم
إبدأ
أكتب_سطر(أ [ ب ]);
إنتهى ؛
إنتهى .
```

A2.1.8 Les procédures et les fonctions

On peut regrouper un ensemble d'instructions sous un même nom. On forme alors un sous programme ou procédure. On utilise les procédures :

- ✓ chaque fois qu'une même suite d'instructions doit être répétée plusieurs fois dans un programme,
- ✓ quand une suite d'instruction forme une action globale. Le programme est alors plus clair et les erreurs plus facilement détectables.

Pour pouvoir utiliser une procédure, il faut d'abord la déclarer.

Structure d'une entité de programme (routine) :

Entête

déclaration des : constantes , variables;

إجراء nom_procédure (paramètres globales)؛

déclaration des : constantes , variables;

إبدأ instructions إنتهى .

L'appel de la procédure se faite comme suit :

مناداة_ إجراء nom_procedure (paramètres) ؛

Tout ce qui a été dit pour les procédures s'applique également aux fonctions. La différence avec une procédure est qu'une fonction renvoie un résultat. L'entête est du type :

وظيفة nom_fonction (paramètres globales) : type_de_la_fonction ؛

La fonction étant le type du résultat retourné. On retourne le résultat par :

مناداة_ وظيفة nom_fonction (paramètres) =: identificateur ؛

EXEMPLE 9

```
برنامج ترتيب ؛
متغيرات
أ ، ب ، ن : حقيقي ؛
وظيفة أكبر (ج ، د : حقيقي) : حقيقي ؛
إبدأ
إذاكان ( ج >=د ) فإن
إبدأ
أكبر:= ج؛
إنتهى ؛
```

```
وإلا
إبدأ
أكبر:= د؛
إنتهى ؛
إنتهى ؛
إبدأ
أكتب_سطر("أدخل قيمتين : " )؛
إقرأ_سطر( أ ، ب ) ؛
مناداة _ وظيفة
ن:= أكبر (أ ، ب)  ؛
أكتب_سطر("أكبر قيمة هي :  " ، ن )؛
إنتهى .
```

WEBOGRAPHIES

[S1] http://www.info.univ-angers.fr/~gh/hilapr/hilapr.htm

[S2] http://www.العربية للكمبيوتر والأنترنت الموسوعة .com

[S3] www.lirmm.fr/~ferber/Compilation/compil1.htm

[S4] http://www.developpez.com/general/cours/PolyCompil.pdf

[S5] http://www.jcolibri.com/index.html

BIBLIOGRAPHIES

[Aho.89] A.Aho, R.Sethi, J.Ullman . compilateur : principles, teckniques et outils 1989.

[Pau04] Ckristine.Paulin, Morning.Cours de compilation université paris sud 2004-2005

[Gar.01] Henri Garreta, Techniques et outils pour la compilation, Faculté des Sciences de Luminy - Université de la Méditerranée Janvier 2001.

[Jerz.98] Jerzy Karczmarczuk, implantation des langages de programmation – compilateurs et interprètes, Licence d'Informatique, Caen 1997/1998.

[Phil.93]Philippe Marquet, Travaux dirigés et travaux Pratiques pour la construction d'un petit compilateur,1993.

[Dria.88]H.Drias, Compilation cours et exercices, Institut d'informatique, 1988.

[G.Dev]Guide du développeur, Borland Delphi 7 pour Windows.

[Sup04]Compilation et théorie des langages, université de Bretagne Occidentale, IUP Ingénierie Informatique.2004