**République Algérienne Démocratique et Populaire**

Ministère de l’Enseignement Supérieur et de la Recherche Scientifique

**Université des Sciences et de la Technologie Houari Boumediene**

**Spécialité :**

3 ème année

Sécurité Informatique

Projet Compilation :

**Réalisé par:**

1. DOB Serine
2. GUENDOUL Hayat
3. BENTOURI Hasnaa
4. BENHADIA Nour

Table de matiere :

**Table des figures**

# Introduction

La programmation a débuté avec l'utilisation du code binaire (0 et 1), qui constitue le langage natif des machines. Cependant, ce type de programmation est particulièrement difficile à implémenter, car il est long et complexe. Pour remédier à ces limitations, des langages deprogrammation de bas niveau, tels que l'assembleur, ont été développés.

L'assembleur, bien qu'étant plus simple à utiliser que le code binaire, présente une contrainte majeure : il nécessite une connaissance approfondie de l'architecture interne de la machine (processeur, registres, etc.). Pour simplifier davantage la tâche des développeurs, des langages dehaut niveau ont été créés (python , java…). Ces langages permettent d'abstraire la complexité de l'architecture machine et rendent la programmation plus accessible.

Cependant, les langages de haut niveau ne sont pas directement compréhensibles par la machine. Ils doivent être traduits en langage machine par des processus tels que l’interprétation, la compilation, ou la traduction. Dans le cadre de ce projet, nous nous intéressons particulièrement au processus de compilation, qui est constitué de plusieurs étapes essentielles pour rendre un programme compréhensible et exécutable par la machine.

# Problématique

Les langages de programmation de haut niveau sont reconnus pour leur simplicité d'utilisation et leur capacité à rendre la programmation plus accessible. Cependant, leur implémentation constitue une tâche particulièrement complexe. Quels sont les détails techniques qui entrent en jeu dans l'implémentation de ces langages ? Comment sont-ils développés, et quelles étapes précises faut-il suivre pour créer un nouveau langage de programmation de haut niveau ?

# Objectif

Dans ce projet, notre objectif est de développer **MinING**, un langage de programmation de haut niveau doté d'une syntaxe spécifique. Pour atteindre cet objectif, nous suivrons un processus structuré comprenant les étapes suivantes :

* **Analyse lexicale** : identification et classification des unités lexicales.
* **Traduction dirigée par la syntaxe** : une étape qui combine l'analyse syntaxique et l'analyse sémantique pour vérifier la structure du programme tout en intégrant les aspects sémantiques.
* **Génération des quadraplets** : création d'une représentation intermédiaire structurée.
* **Génération de code intermédiaire** : production d'un code prêt pour les étapes ultérieures de compilation.

# **Chapitre 1 : Analyse Lexicale**

Dans le cadre de ce projet, la phase de l’analyseur lexical est développée à l'aide de l'outil Flex. L'analyse lexicale joue un rôle crucial dans le processus de compilation car elle permet de prendre en charge le texte source et de le découper en unités de sens, appelées tokens. Ces tokens sont créés pour être utilisés par l'analyseur syntaxique.

L'objectif principal de cet analyseur lexical est de reconnaître différentes entités lexicales telles que les mots-clés, les identificateurs, les opérateurs, les constantes numériques, etc. Chaque entité est associée à un tokenspécifique qui facilite l'analyse syntaxique du code.

De plus, l’analyseur lexical prend en charge la détection des erreurs lexicales, en rejetant toute entité non définie. Cela permet une localisation rapide des erreurs dans le code source.

Flex défini l’entité lexicale à reconnaitre à l’aide d’expressions régulières. Pour notre projet nous avons générer un analyseur qui reconnais :

|  |  |  |
| --- | --- | --- |
| **Entité lexicale** | **Expression Régulière** | **Symbol** |
| Déclaration entier | INTEGER | INTEGER |
| Déclaration réel | FLOAT | FLOAT |
| Caractère(declaration) | CHAR | CHAR |
| Constante | CONST | CONST |
| Si | IF | IF |
| Sinon | ELSE | ELSE |
| Pour | FOR | FOR |
| Scanf | READ | READ |
| Printf | WRITE | WRITE |
| Identifiant | [A-Z][a-z0-9]{0,7} | IDENTIFIER |
| Nombre positive | [+]?[0-9]+ | NUMBERINTPOS |
| Nombre positive | [+]?[0-9]+\.[0-9]+ | NUMBERFLOATPOS |
| Nombre négative | "-"[0-9]+ | NUMBERINTNEG |
| Nombre float négative | "-"[0-9]+\.[0-9]+ | NUMBERFLOATNEG |
| Caractère | [A-Za-z] | caractere |
| Op ET | && | AND |
| OP OU | \|\| | OR |
| OP NON | ! | NOT |
| OP Egale | == | EQUAL |
| OP Différent | != | NEQ |
| Supérieur ou égale | >= | GTE |
| Inférieur ou égale | <= | LTE |
| Supérieur strict | > | GT |
| Inférieur strict | < | LT |
| Addition | + | PLUS |
| Soustraction | \_ | MINUS |
| Multiplication | \* | MULT |
| Division | / | DIV |
| { | { | LBRACE |
| } | } | RBRACE |
| ) | ( | LPAREN |
| ) | ) | RPAREN |
| [ | [ | LBRACKET |
| ] | ] | RBRACKET |
| ; | ; | SEMICOLON |
| : | \ : | COLON |
| Vergule | , | COMMA |
| Affectation | = | ASSING |

# Table de symbole :

Dans la phase d’analyse lexicale nous avons créé la table des symboles avec ses fonctions. Nous avons utilisé trois tables statiques pour chaque type d’entité lexicale : Une table pour les mots-clés, Une table pour les identificateurs,Une table pour les séparateurs.

La table des symboles comporte cinq fonctions : initialisation, recherche, insertion, affichage et modification.

## Initialisation :

La fonction initialiserTable est appelée pour initialiser la table de symboles qui sera utilisée lors de la compilation d’un programme. Chaque entrée de cette table correspond à un symbole qu’il soit une variable, une constante, un mot-clé et ainsi de suite et regroupe des informations telles que : l’état, le nom de l’espace, le type, la valeur, la portée, l’offset ainsi que d’autres attributs thématiques qui interviennent dans la manipulation du symbole. chaque entrée est marquée comme libre, avec un nom et un type vides, une valeur initialisée à 0.0, une portée globale et un offset de -1, indiquant qu'aucune affectation n'a encore été faite à cette entrée. De plus, définir des indicateurs tels que isArray et isConst à 0, ce qui signifie que l’entrée n’est ni un tableau ni une constante.

## Recherche :

 Cette fonction a été créée pour rechercher un élément dans la table des symboles. Cela est très utile lors de l'analyse sémantique, par exemple pour vérifier si les attributs de type sont respectés ou si une variable a été déclarée au préalable.

## Insertion :

Insérer les nouveaux identifiants dans la table des symboles uniquement s’ils ne sont pas déjà présents.

## Modification :

Pour modifier les valeurs des identifiants dans la table de symboles en cas de mise à jour dans le code source, cette fonction vérifie d'abord si l'identifiant n'est pas une constante, car une constante ne peut pas être modifiée. Ensuite, elle s'assure que l'identifiant existe déjà dans la table en utilisant la fonction de recherche.

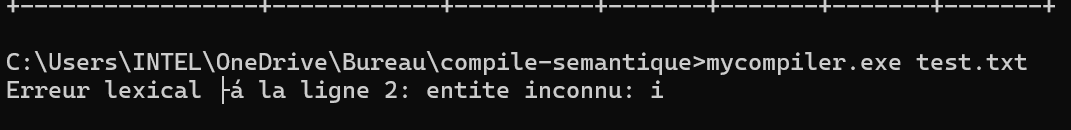
# **Chapitre 2 : Analyse Syntaxique**

L'analyse syntaxique consiste à mettre en évidence la structure d'un texte à l’aide d’un ensemble de règle de syntaxe formant une grammaire. Cette étape suit l’analyse lexicale et utilise les entités produites par cette dernière. Pour notre projet nous avons réalisé un analyseur syntaxique (perser) à l’aide de l’outil Bison qui permet de générer des analyseurs avec des grammaires.

Exemple1 :

VAR\_GLOBAL {

int a, b; float c;

} 

Exemple2 :

VAR\_GLOBAL {

INTEGER A;

CONST FLOAT Pi = 3.14;

INTEGER Q;

FLOAT Y;

CONST INTEGER Max= 100;

}

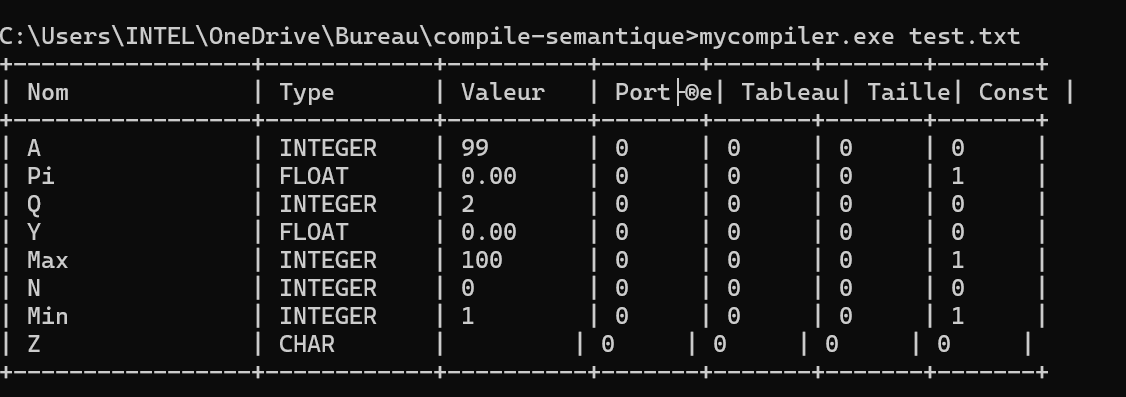
DECLARATION { %% local variables

INTEGER N;

CONST INTEGER Min = 1;

CHAR Z;

}



# **Chapitre 3 : Analyse Semantique**