UNIVERSITÉ DE BÉJAÏA

Faculté des Sciences Exactes
Département d'Informatique

Rapport de Mini-Projet

Module : Compilation

Mini-Compilateur Langage C avec instruction while

Réalisé par :
Malak Bensafia
Groupe A2

Encadré par :
Mme. Nadia Tassoult

Décembre 2025

Table des matières

1 Introduction **3**

2 Grammaire Utilisée **3**
2.1 Explication des règles 3
2.2 Exemples 3

3 Analyseur Lexical **4**
3.1 Résumé du fonctionnement 4
3.2 Méthode de travail 4
3.3 Exemple de reconnaissance 4
3.4 Mots-clés reconnus 4
3.5 Éléments détectés 4

4 Analyseur Syntaxique **5**
4.1 Méthode utilisée 5
4.2 Fonctions principales 5

5 Tests et Résultats **5**
5.1 Test avec boucle while 5

6 Structure du Projet **5**

7 Conclusion **6**

1 Introduction

Ce projet implémente un mini-compilateur en Java qui analyse du code source en langage C. Le programme reconnaît plusieurs éléments de base du langage C et vérifie leur validité syntaxique.

Fonctionnalités principales :

— Analyse lexicale : découpe le code en unités
— Analyse syntaxique : vérifie la structure
— Détection d'erreurs : messages clairs
— Gestion de la boucle `while`

2 Grammaire Utilisée

```
1.  S → A #
2.  A → B A | epsilon
3.  B → E ; | H ; | C | D | F | G | { A } | ;
4.  E → int id | int id = J
5.  H → id = J
6.  C → while ( I ) { A }
7.  D → if ( I ) B | if ( I ) B else B
8.  F → do { A } while ( I ) ;
9.  G → for ( J ; I ; J ) { A }
10. I → J | J OPC J
11. J → K | J + K | J - K
12. K → L | K * L | K / L
13. L → id | nombre | ( J )
14. OPC → < | > | = | !=
```

2.1 Explication des règles

— **Règle 1** : Programme terminé par #
— **Règle 2** : Suite d'instructions (peut être vide "epsilon")
— **Règle 3** : Une instruction peut être déclaration (E ;), affectation (H ;), structure de contrôle, bloc, ou simplement point-virgule (;)
— **Règles 4-5** : Déclarations de variables et affectations
— **Règles 6-9** : Structures de contrôle (while, if, do, for)
— **Règles 10-13** : Expressions arithmétiques et logiques
— **Règle 14** : Opérateurs de comparaison

2.2 Exemples

— `B → E ;` : `int x;` (déclaration avec point-virgule)
— `B → H ;` : `x = 5;` (affectation avec point-virgule)
— `B → ;` : Un simple point-virgule
— `C → while ( I ) { A }` : `while (x > 0) { x = x - 1; }`

3 Analyseur Lexical

3.1 Résumé du fonctionnement

L'analyseur lexical fonctionne **instruction par instruction** en parcourant le code caractère par caractère. Il reconnaît les mots-clés, identificateurs, nombres, opérateurs et symboles spéciaux du langage C.

3.2 Méthode de travail

— Lecture séquentielle du code source
— Comparaison caractère par caractère
— Utilisation de conditions if/else pour identifier les tokens
— Production d'une liste de tokens classés

3.3 Exemple de reconnaissance

Pour reconnaître le mot-clé `while` :

```
if (tc == 'w') {
    i++; tc = s.charAt(i);
    if (tc == 'h') {
        i++; tc = s.charAt(i);
        if (tc == 'i') {
            i++; tc = s.charAt(i);
            if (tc == 'l') {
                i++; tc = s.charAt(i);
                if (tc == 'e') {
                    System.out.println("MOT_CLE : while");
                }
            }
        }
    }
}
```

3.4 Mots-clés reconnus

— Mots-clés de base : `while, if, else, do, for, int`
— Types : `float, char, void`
— Contrôle : `return, break`

3.5 Éléments détectés

— Identificateurs (variables)
— Nombres entiers
— Opérateurs : `+ - * / < > = !=`
— Parenthèses et accolades
— Point-virgule

4 Analyseur Syntaxique

4.1 Méthode utilisée

— Descente récursive
— Une fonction par règle de grammaire
— Vérification pas à pas

4.2 Fonctions principales

— `S()` - Programme principal
— `A()` - Suite d'instructions (avec epsilon)
— `B()` - Instruction simple (peut être juste un point-virgule)
— `C()` - Traitement du while
— `I()` - Conditions
— `J(), K(), L()` - Expressions

5 Tests et Résultats

5.1 Test avec boucle while

```
Entrée : while (x > 0) { x = x - 1; }

Résultat lexical :
MOT_CLE : while
PARENTHESE : (
IDENTIFICATEUR : x
OPERATEUR : >
NOMBRE : 0
PARENTHESE : )
ACCOLADE : {
IDENTIFICATEUR : x
OPERATEUR : =
IDENTIFICATEUR : x
OPERATEUR : -
NOMBRE : 1
POINT_VIRGULE : ;
ACCOLADE : }
FIN

Résultat syntaxique :
Programme syntaxiquement correct
```

6 Structure du Projet

```
MiniCompilateur/
 LexicalProjet.java
 SyntaxiqueProjet.java
 Main.java
```

7 Conclusion

Ce projet permet de comprendre les bases de la compilation. Le compilateur fonctionne correctement pour du code C simple avec la boucle while. L'analyse lexicale et syntaxique sont bien réalisées, et le programme détecte les erreurs efficacement.