Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique

Master 1 :IL-IV Année Universitaire : 2024/2025

Module Compilation

Titre du Projet : Compilateur d'une version simplifiée de Java avec FLEX et BISON

Objectif :

Développer un compilateur capable de traiter une version simplifiée du langage **Java**, incluant les déclarations de classes, les méthodes, les variables, les opérations de base, les structures de contrôle, et les exceptions.

Outils :

- **FLEX** pour l'analyse lexicale.
- **BISON** pour l'analyse syntaxique et sémantique.

Description du Langage Java Simplifié :

Ce langage simplifié de Java supportera les éléments suivants :

1. **Types de base** : int, float, double, char, boolean.
2. **Variables** : Déclaration et initialisation avec les types de base.
3. **Opérations arithmétiques** : +, -, *, /, %.
4. **Opérations logiques** : &&, ||, !.
5. **Structures de contrôle** : if, else if, else, switch.
6. **Boucles** : for, while, do-while.
7. **Classes et Objets** : Déclaration simple de classes avec des attributs et des méthodes.
8. **Fonctions/Méthodes** : Méthodes simples sans surcharge.
9. **Gestion des Exceptions** : Prise en charge de try, catch, et finally (simplifié).
10. **Entrée/Sortie** : Utilisation de System.out.println() pour l'affichage.

Phases du Projet :

1. **Analyse Lexicale :**
    - Utilisation de **FLEX** pour définir un analyseur lexical capable de reconnaître les mots-clés Java (tels que class, int, if, else, for, return), les identificateurs (noms de variables, de méthodes), les opérateurs, et les séparateurs (;, {}, etc.).
2. **Analyse Syntaxique :**
    - Utilisation de **BISON** pour créer un analyseur syntaxique qui suit la grammaire simplifiée de Java. Cela inclura :
        - Les classes avec des méthodes.
        - Les déclarations de variables et leur initialisation.
        - Les instructions conditionnelles (if, else) et les boucles (for, while).
        - Les méthodes simples avec retour de type et paramètres.
3. **Gestion de la Table de Symboles :**
    - Mise en place d'une **table de symboles** pour gérer les variables locales, les paramètres de méthode, et les attributs de classe.
    - La table de symboles devra gérer la portée des variables à l'intérieur des méthodes et des classes, et s'assurer qu'aucune variable n'est utilisée hors de sa portée.
4. **Analyse Sémantique :**
    - Vérification des types (cohérence entre les types des variables et des opérateurs), gestion des portées, et vérification de la validité des instructions (par exemple, qu'une méthode retourne le bon type).
    - Gestion des erreurs sémantiques comme la redéclaration de variables ou l'incompatibilité des types dans les expressions, etc.
5. **Génération de Code Intermédiaire :**
    - Générer un code intermédiaire, par exemple sous forme de **quadruplet**, qui sera plus facile à traduire en code exécutable dans une étape ultérieure.

6. **Optimisation du Code Intermédiaire :**
    - Appliquer des optimisations sur le code intermédiaire, telles que la réduction de code inutile, l'élimination du code mort, la simplification des expressions constantes, etc.
7. **Génération de Code Machine :**
    - Le code machine doit être généré en **assembleur 8086**.
8. **Traitement des erreurs :**
    - Il est demandé d'afficher les messages d'erreurs adéquats à chaque étape du processus de compilation. Ainsi, lorsqu'une erreur lexicale ou syntaxique est détectée par votre compilateur, elle doit être signalée le plus précisément possible, par sa nature et sa localisation dans le fichier source. On adoptera le format suivant pour cette signalisation :

      Type_ de_ l'erreur, num_ligne, num_colonne , entité qui a généré l'erreur.

9. **Tests et Validation :**
    - Développer un ensemble de programmes Java simplifiés pour tester le bon fonctionnement du compilateur sur les classes, méthodes, boucles, conditions et exceptions. Assurer la robustesse du compilateur à travers une série de tests.