Rapport pour le mini compilateur

1. Ma grammaire : (avec Z)

Z → S EOF

S → while (COND) { INST_LIST }

COND → VARIABLE OPERATEUR VARIABLE | VARIABLE OPERATEUR NOMBRE

INST_LIST → INST INST_LIST | ε

INST → VARIABLE ;

| VARIABLE OPERATEUR VARIABLE; | VARIABLE OPERATEUR NOMBRE ;

| S

2. L'analyseur lexical :

L'analyse lexicale de mon mini-compilateur est basée sur une matrice de transition et des états finaux. Il lit la chaîne de caractères entrée et la décompose en unités lexicales (tokens). Chaque token correspond à une catégorie précise, comme MOT_CLE, MOT_CLE_PERSONNALISE, VARIABLE, IDENTIFICATEUR, NOMBRE, OPERATEUR ou SEPARATEUR. Les commentaires et les espaces sont ignorés ou utilisés uniquement pour séparer les tokens. Chaque token extrait est affiché sous la forme Token: "valeur" Type: CATEGORIE et, en parallèle, tous les tokens ainsi que leurs types sont stockés dans des tableaux afin d'être utilisés par l'analyse syntaxique pour vérifier la structure grammaticale du code.

3. L'analyseur syntaxique :

Chaque règle de la grammaire de mon mini-compilateur correspond à une procédure. L'analyse syntaxique utilise la méthode descendante récursive pour vérifier la justesse de la grammaire en analysant le code entré. Elle se sert des tableaux de tokens et de leurs types produits par l'analyse lexicale pour parcourir et contrôler la structure du code. En cas d'erreurs, l'analyse syntaxique les détecte et les affiche, ce qui permet de savoir pourquoi la chaîne n'est pas acceptée.

4. La structure :

Un projet : Mini_Compilateur.

Un package nommé : Projet_Mini_Compilateur, qui se trouve dans Source Packages.

Ce package a deux classes : AnalyseLexicale avec le main et AnalyseSyntaxique.

5. Test :

Chaine acceptée : while($i=1){ $j=2;}

Chaine non acceptée : while($i=4)