Rapport de Projet : Mini-Compilateur Try/Catch PHP

Roza Oumansour

Groupe B3

Introduction

Ce projet consiste à développer un **mini-compilateur** pour analyser un langage inspiré de PHP, avec une attention particulière à la gestion des erreurs (try-catch). L'objectif est de créer deux composants essentiels : un **analyseur lexical** qui décompose le code source en unités de base (tokens), et un **analyseur syntaxique** qui vérifie si la structure du code respecte une grammaire définie.

Le projet intègre également une **interface graphique** permettant de tester facilement des extraits de code et d'afficher les résultats des analyses ainsi que les erreurs détectées.

À travers cette réalisation, nous mettons en pratique les principes fondamentaux de la compilation : analyse lexicale, analyse syntaxique et gestion d'erreurs, dans un contexte simplifié mais fonctionnel.

1. Grammaire Choisie

Z → S FIN_FICHIER

S → Instruction S | ε

Instruction → VARIABLE (= Expression | ++ | --) ;

| try { S } catch (IdentifiantException) { S }

| NOM PRENOM ;

| InstructionIgnoree

Expression → Terme { (+|-|*|/|%) Terme }

Terme → NOMBRE | CHAINE | VARIABLE | IDENTIFIANT | (Expression)

2. Analyseur Lexique (Lexical Analyzer)

2.1 Caractéristiques principales

- Langage cible : Sous-ensemble de PHP avec gestion d'exceptions

- Classes implémentées : `AnalyseurLexical`, `Token`, `TokenType`

- Méthode de balayage : Analyse caractère par caractère avec automate à états finis

2.2 Types de tokens reconnus

- Tokens spéciaux : `FIN_FICHIER`, `IDENTIFIANT`, `VARIABLE` (préfixe $)

- Mots-clés PHP : `TRY`, `CATCH`, `EXCEPTION`, `IF`, `ELSE`, `FOR`, `WHILE`

- Mots-clés personnalisés** : `NOM`, `PRENOM` (Oumansour, Roza)

- Constantes : `NOMBRE`, `CHAINE` (entre guillemets)

- Symboles : Parenthèses, accolades, point-virgule

- Opérateurs : Arithmétiques (`+`, `-`, `*`, `/`, `%`, `++`, `--`), comparaison (`==`, `!=`, `<`, `>`, `<=`, `>=`), logiques (`&&`, `||`, `!`)

2.3 Algorithme de scan

1. Lecture caractère par caractère

2. Classification selon le premier caractère

3. Construction des lexèmes pour les identifiants, nombres et chaînes

4. Gestion des commentaires ignorés (non implémenté dans cette version)

5. Suivi des numéros de ligne

2.4 Points forts

- Gestion des opérateurs multi-caractères (`==`, `++`, `&&`, etc.)

- Reconnaissance des variables PHP (`$variable`)

- Traitement correct des chaînes de caractères

3. Analyseur Syntaxique (Syntax Analyzer)

3.1 Architecture

- Type : Analyseur descendant récursif (Recursive Descent Parser)

- Stratégie d'erreur : Récupération par synchronisation sur `;`, `}` ou `FIN_FICHIER`

- Gestion d'erreurs : Messages détaillés avec numéro de ligne et token problématique

3.2 Fonctions principales

- Z() : Point d'entrée, vérifie `S FIN_FICHIER`

- S() : Séquence d'instructions

- Instruction() : Délégation selon le token courant

- TryCatch() : Analyse des blocs try-catch

- Expression() : Évaluations arithmétiques et logiques

3.3 Stratégie de récupération d'erreurs

1. Détection d'une incompatibilité

2. Enregistrement du message d'erreur

3. Avance jusqu'au prochain point de synchronisation

4. Structure du Projet

Arborescence des fichiers

```
MiniCompilateur/
│
├── src/
│   ├── Mainn.java              # Interface graphique principale
│   ├── AnalyseurLexical.java       # Analyseur lexical
│   ├── AnalyseurSyntaxique.java    # Analyseur syntaxique
│   ├── Token.java              # Classe Token
│   └── TokenType.java          # Énumération des types de tokens
│
├── README.md                   # Documentation
└── rapport.md                  # Ce rapport
```

4.1 Description des classes

1. `Mainn` : Interface Swing avec deux zones de texte et boutons d'analyse

2. `AnalyseurLexical`: Transforme le code source en tokens

3. `AnalyseurSyntaxique` : Vérifie la structure grammaticale

4. `Token` : Représente un token avec type, texte et ligne

5. `TokenType`: Énumération exhaustive des tokens possibles

4.2 Flux d'exécution

Code source → AnalyseurLexical → Liste de Tokens → AnalyseurSyntaxique → Résultat/Erreurs

↑ ↑

Interface GUI Affichage résultats

5. Cas de Test

5.1 Tests syntaxiques valides

Test 1 : Affectation simple

```
$x = 10;
$y = $x + 5;
```

Test 2 : Bloc try-catch**

```
try {
   $result = 100 / 0;
} catch (Exception $e) {
   echo "Erreur divisée";
}
```

Test 3 : Signature personnalisée

```
Oumansour Roza;
```

Test 4 : Instructions combinées

```
$x = 5;
$y = (10 + $x) * 2;
try {
   if ($y > 20) {
      $z = $y / 2;
   }
} catch (Exception ex) {
```

```
   $error = "Problème";
}
Oumansour Roza;
```

5.2 Tests avec erreurs syntaxiques

Test 5 : Point-virgule manquant

```
$x = 10  // ERREUR: ';' attendu
$y = 20;
```

Test 6 : Parenthèse non fermée

```
$z = (10 + 5 * 2;  // ERREUR: ')' attendu
```

Test6 7 : Structure try-catch incorrecte

```
try {
   $a = 1;
catch Exception  // ERREUR: '}' et '(' manquants
```

Test 8 : Signature incomplète

```
Oumansour;  // ERREUR: PRENOM attendu
```

5.3 Tests limites

est 9 : Code vide

```
// Doit accepter sans erreur
```

Test 10 : Instructions ignorées

```
php
```

```
if (true) { echo "test"; }

while (false) { /* rien */ }

// Doit être ignoré sans erreur
```

Test 11 : Expressions complexes

```
$x = ((10 + 5) * (20 - 3)) / 2;

$y = $x % 3 + 1;
```

6. Observations :

6.1 Points forts

1. Interface graphique intuitive
2. Messages d'erreur détaillés
3. Gestion robuste des erreurs avec récupération
4. Reconnaissance des motifs PHP courants

7. Conclusion

Ce projet implémente un mini-compilateur pour un sous-ensemble de PHP avec une attention particulière sur la gestion des exceptions via les blocs try-catch. L'analyseur lexical reconnaît les constructions principales du PHP, tandis que l'analyseur syntaxique vérifie la conformité à une grammaire définie. L'interface graphique permet une interaction facile avec l'outil pour l'analyse de code.

Le système démontre les principes fondamentaux de la compilation : analyse lexicale, analyse syntaxique, et gestion d'erreurs. Bien que limité en scope, il forme une base solide pour des extensions futures vers un compilateur plus complet.