Baloul Tachfin

Groupe : A1

Compilation

Introduction

L'objectif de ce projet est de réaliser un mini-compilateur pour une construction du langage JavaScript : l'instruction do { … } while (condition);.

Le compilateur est composé de deux parties principales :

• un analyseur lexical qui transforme le texte source en une suite de tokens ;

• un analyseur syntaxique qui vérifie que cette suite de tokens respecte une grammaire définie pour le sous-langage choisi.

1• Analyseur Lexicale

Il prend en entrée une chaîne contenant le code source, à laquelle on ajoute un caractère de fin #, et parcourt le texte caractère par caractère en maintenant :

• position : l'indice du caractère courant

• ligne : le numéro de ligne (pour les messages d'erreur)

• une liste erreursLex qui stocke les erreurs lexicales détectées.

Pour reconnaitre les id et les mots clés on utilise ici un AFD avec une colonne pour les lettres et '_', une pour les chiffres et une pour les autres caractères. Et on a la matrice suivante :

Etat	Lettre_	Chiffre	Autres
0	1	-1	-1
1	1	1	-1

Fonctionnement :

• en état 0, seule une lettre ou _ peut commencer un identificateur → passage à l'état 1

• en état 1, on accepte lettres et chiffres → l'identificateur se prolonge

• dès qu'un caractère de classe « autre » arrive, l'automate s'arrête et l'identificateur est terminé.

On a plusieurs Tokens reconnu

Pour les mots clés il y'a :

DO, WHILE,

LET, VAR, CONST, SUPER, NULL, THIS, TYPEOF,

True, False,

IF, ELSE, FOR, RETURN, FUNCTION,

BREAK, SWITCH, CASE, DEFAULT,TACHFIN,BALOUL,

Pour les délimitateurs on a :

LPAR, RPAR, ( )

LACCO, RACCO, { }

POINTV, ;

Pour les opérateurs on a :

AFFECT, =

INF, SUP, < >

INFEG, SUPEG, <= >=

EGAL, ==

INCR, DECR, ++ --

Pour les id et nombres on a :

ID,NOMBRE

Pour les erreurs lexicales :

ERREUR

Et enfin pour la fin du fichier :

EOF

En cas de caractère non reconnu (par exemple un symbole isolé non supporté), le lexer :

• ajoute un message descriptif dans erreursLex

• retourne un token de type ERREUR

2•Analyseur syntaxique

L'analyseur syntaxique est un analyseur descendant récursif qui consomme les tokens fournis par le lexer et vérifie qu'ils respectent la grammaire suivante :

Programme → DoWhile EOF

DoWhile → DO Bloc WHILE (Condition) ;

Bloc → { ListeInst }

ListeInst → Instruction ListeInst | ε

Instruction →

DoWhile

| Déclaration

| Affectation

| Incrémentation

| Décrémentation

Declaration → (LET | VAR | CONST) Id (= Expression) ? ;

Affectation → Id= Expression ;

Incrémentation → Id++ ;

Décrémentation → Id -- ;

Id → ID

Condition → Expression Operateur Expression

| Expression

Operateur → < | <= | > | >= | ==

Expression → Ident | NOMBRE | True | False

Chaque non terminal est implementé par une methode Java.

Les erreurs syntaxiques sont mémorisées dans une liste, en indiquant la ligne, le message, et le token trouvé, et un mécanisme de récupération (panic mode) permet d'éviter les cascades d'erreurs.

3•Arborescence de dossier et de fichier

DOWHILE/

```
│
├── .git/
├── exécutable/
│       └──A1_Baloul_Tachfin.jar
│
├── src/
│   └── main/
│       └── java/
│           └── com/
│               └── mycompany/
│                   └── dowhile/
```

```
│                   ├── AnalyseurLexDOWHILE.java
│                   ├── AnalyseurSyntaxiqueDOWHILE.java
│                   ├── Token.java
│                   ├── TokenType.java
│                   └── DOWHILE.java (main)
│
├── target/
│   ├── classes/
│   └── DOWHILE-1.0-SNAPSHOT.jar
│
├── .gitignore
│
└── pom.xml
```

4•Les cas de tests

DoWhile simple :

```
do {
   let x = 5;
   x++;
   var y = 10;
   y--;
} while (x < y);
```

En utilisant le nom et prenom :

```
do {
    tachfin = 1;
    baloul = 2;
    baloul--;
} while (tachfin < baloul);
```

Conditions booléenne simple :

```
do {
    const ok = true;
} while (false);
```

Avec commentaires :

```
// Boucle de test
do {
    let x = 1;   // initialisation
    x++;
    /* décrémentation inutile
       y--; */
} while (x < 5);
```

Operateur non supporté :

```
do {
    let x = 1;
```

```
    x = x + 1;
} while (x < 5);
```

Caractère illégale :

```
do {
    let x = 5;
    x = 3 @ 2;
} while (x < 10);
```

Point-virgule manquant :

```
do {
    let x = 5
    x++;
} while (x < 10);
```

Erreur dans la condition :

```
do {
    let a = 2;
} while (a < );
```

Mal écriture de while :

```
do {
    var x = 1;
} whil (x < 2);
```

DoWhile incomplet :

```
do {
    let a = 1;
}
```