CS444 - Projet Compilation Documentation

Les messages d'erreurs

Pour la gestion des erreurs nous avons à disposition ErreurContext.java, ErreurInterneVerif.java et ErreurReglesTypage.java.

Pour la suite nous avons préferé gerer les erreurs avec la classe enum ErreurContext et non par la classe ErreurReglesTypage. En effet, cela nous permet de différencier les erreurs levées et d'afficher des messages spécifiques. D'autre part, si la passe 1 a été correctement faite, il ne devrait pas avoir d'ErreurInterneVerif.

Voici les erreurs contextuelles ainsi que leur description et le message affiché :

- ErreurVariableRedeclaree :
Description : Lors de la déclaration des variables, si une variable a le meme nom qu'une variable déjà déclarée
Message : Erreur contextuelle : Variable non déclarée
Variable [nom_variable] non déclarée ... ligne [num_ligne]

- ErreurVariableNonDeclaree
Description : Si on utilise une variable qui n'a pas été déclarée au préalable
Message : Erreur contextuelle : Variable non déclarée
Variable [nom_variable] non déclarée ... ligne [num_ligne]

- ErreurTypageNonCompatible
Description : Lors d'une affectation, une operation binaire ou une operation unaire et que le type attendu n'est pas le bon.
Message : Erreur contextuelle : Opération non compatible :
Opération Affect : Types [type1] et [type2] incompatibles ... ligne [num_ligne]

- ErreurTypeIndefini
Description : Lors d'une déclaration d'une variable, on lui donne un type indefini
Message : Erreur contextuelle : Typeindefini
Le type[type_indefini] est indefini dans l'environnement ... ligne [num_ligne]

- ErreurIdentNomReserve
Description : Lors d'une utilisation d'un mot reservé
Message : Erreur contextuelle : Identificateur déclaré avec un nom réservé
[mot_reservé] est un nom réservé ... ligne [num_ligne]

- ErreurNonRepertoriee
Description : Aucune des erreurs précédentes, normalement on utilise jamais cette erreur.
Message : Erreur contextuelle : Erreur non repertoriee

La méthodologie de test

L'objectif principale de cette base de test est de tester notre compilateur sur des régles sémantiques décrites dans le fichier context.txt.

Pour ce faire, nous avons respecter certaines conditions afin de couvrir la majeur partie de règles.

— Bien décomposer les problèmes en écrivant des méthodes COURTES.
— Factoriser les éléments communs (Eviter les tests testant les mêmes régles sémantiques)
— Documenter chaque test à l'aide d'un commentaire avant le code testé

On écrira des programmes JCas valides et invalides sémantiquement.

–> Dans le cas valide :
Le programme s'éxecute correctement, on vérifie que l'arbre est correctement décoré.

–> Dans le cas invalide :
- On vérifie que le message d'erreur est pertinent.

Une fois le fichier *context.txt* assimilé, nous avons réalisé une liste de tests exhaustive suivant chacune des régles définies.

Pour chaques régles, nous avons écrits des tests valides sémantiquement, et des tests invalides afin de couvrir le plus de cas possibles.