Université de Lorraine
Licence Informatique - Semestre 6

UE Compilation - Zoot Version 0

L'objectif de cette séance est la construction complète du compilateur **zoot** Version 0.
Seul l'un des deux membres du binôme doit faire un dépôt de l'archive exécutable **zoot.jar** sur Arche **avant la fin de la séance à 10h00.**

A. JFlex et JavaCup

Avant toute chose, il faut recopier les deux archives JFlex et JavaCup fournies sur Arche. Vous pouvez les ranger là où bon vous semble.

B. Mise en place de l'environnement de travail

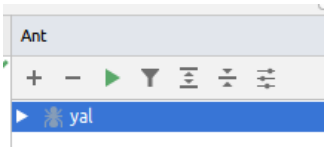
1. Pour faciliter le travail en équipe et le suivi d'avancement des itérations, vous devez utiliser le dépôt **gitlab** qui vous est créé pour ce projet. Rendez-vous sur la page https://gitlab.univ-lorraine.fr/ pour trouver l'url de votre projet.

   Ce dépôt est régulièrement consulté par les enseignants pour évaluer l'usage qui en est fait. À chaque dépôt d'archive **zoot.jar** sur arche,, vous devez ajouter un tag (**zoot0** ou **zoot1** ou ….) qui permet de retrouver la version correspondante sur la forge gitlab.

2. Créez un nouveau projet IntelliJ à partir du dépôt **zoot** (peu importe son nom). La souche de ce projet contient le package **zoot** avec les sources du compilateur (fichiers sources pour les générateurs d'analyses lexicale/syntaxique, classes de l'arbre abstrait et programme principal).
   Adaptez le fichier **build.xml** en fonction des chemins d'accès aux deux archives JFlex et JavaCup.

   Intégrez les deux archives JFlex et JavaCup dans les librairies du projet.

3. Pour lancer la création des deux analyseurs et la compilation de l'ensemble des classes, exécutez le fichier **build.xml** (clic droit sur le fichier et sélection de "**Add as Ant Build File**"). Dans la fenêtre **ant** ouverte, sélectionnez le mot "zootl" pour faire apparaître la flèche verte sur laquelle cliquer.

   Vérifiez que l'exécution de **ant** se passe sans souci.

   À noter que la modification des fichiers **Grammaire.cup** et **UnitesLexicales.jflex** n'entraîne pas automatiquement la reconstruction des classes Java correspondantes : il faut systématiquement exécuter à nouveau le fichier **build.xml**.

C. Créer une archive exécutable

Créer une archive exécutable **zoot.jar** pour ce premier noyau du langage.

Testez l'exécution de cette archive sur un fichier source **zoot0**, dans une fenêtre terminal. L'exécution déclenche l'exception **UnsupportedOperationException**, lors d'un appel à **toMips**().

Il est opportun de créer un répertoire destiné à contenir tous les sources **zoot0** utilisés pour faire les tests. Ces fichiers de tests peuvent être partagés dans un pot commun créé à cette occasion sur Arche.

D. Compléter les classes existantes

Consultez le diagramme de classes du code fourni.

Le programme principal prévoit la réalisation de l'analyse syntaxique d'un texte source (appel de **parse()**), puis l'analyse sémantique de l'arbre abstrait construit (appel de **verifier()**), puis la génération de code de l'arbre abstrait vérifié (appel de **toMips()**).

Les fichiers sources des analyseurs (lexical et syntaxique) sont fournis, il n'y a pas lieu de les modifier pour cette version de zoot. Les fonctions **verifier** et **toMips** ne sont pas écrites. La fonction **verifier** n'a pas lieu d'être en **zoot0**. Il faut donc simplement compléter la fonction **toMips** dans les classes fournies.

Il est judicieux de commencer tout de suite à écrire des classes de tests JUnit.

E. Consignes à respecter impérativement (avant 10h00)

- Push du projet sur gitlab avec le tag **zootl0**.
- Dépôt sur Arche de l'archive **zoot.jar**, cohérente avec le dépôt gitlab.

F. Préparation de zoot1

Le noyau **zoot0** étant terminé, il faut commencer le noyau **zoot1** sans tarder.

D'ici le prochain TD, vous devez :

- modifier le fichier **UnitesLexicales.jflex** pour intégrer les commentaires et les nouvelles unités lexicales **zoot1** ;
- modifier le fichier **Grammaire.cup** pour intégrer les règles de grammaire de **zoot1** (et rien d'autre) avec le code Java qui complète l'arbre abstrait des instructions. Les déclarations seront analysées par l'analyseur syntaxique, mais momentanément pas utilisées ;
- compléter l'ensemble des classes avec les nouvelles classes d'arbre abstrait.

Pour vous aider, vous trouverez sur Arche un document présentant JFlex et JavaCup.

Vous garderez pour la semaine suivante (après les explications données au prochain TD) la gestion des déclarations (c'est-à-dire la création de la table des symboles), l'écriture des fonctions **vérifier** et **toMips**.

Pensez à maintenir à jour le diagramme de classes et à l'intégrer au dépôt gitlab.

Vous connaissez le planning des différentes versions, ne vous laissez pas déborder. Si besoin, posez des questions sur le forum du projet prévu à cet effet.