TP1 – Grammaire et analyseur syntaxique
hivers 2019

Chargé de laboratoire : Félix Brunet (felix.brunet@polymtl.ca)

1. Objectifs

- Se familiariser avec JavaCC
- Utiliser un analyseur lexical
- Construire un analyseur syntaxique descendant à l'aide de JavaCC

2. Travail à faire

JavaCC (Java Compiler Compiler)[1] est utilisé afin de générer les analyseurs lexical et syntaxique en Java à partir de règles décrites dans un fichier `.jjt`. Le fichier fourni (`Template.jjt`) est divisé en deux sections : une pour l'analyse lexicale et une pour l'analyse syntaxique. L'analyseur lexical permet de séparer le programme fourni en entrée en jetons (« tokens »). Ces jetons sont généralement les mots-clés, les opérateurs, les identificateurs, les caractères spéciaux, etc. définis dans le langage.

L'analyseur syntaxique permet quant à lui de déterminer la validité du programme donné en entrée. Il analyse les jetons retournés par l'analyseur lexical et vérifie si ces derniers respectent les règles définies dans la grammaire. La grammaire définit la syntaxe du langage – l'analyseur syntaxique permet donc de vérifier que la suite de jetons qui constitue le programme en entrée respecte bien la syntaxe du langage.

La grammaire JavaCC qui vous est fournie décrit un langage permettant d'assigner des valeurs à des variables et d'effectuer des opérations arithmétiques élémentaires et d'exécuter certaines fonctions mathématiques.

Modifiez la grammaire JavaCC de façon à ce qu'elles reconnaissent toutes les structures ci-dessous en terminant le corps des fonctions identifier et en ajoutant au besoin de nouvelles fonctions. Le dossier « test-suite/CodeGenTest/data » ainsi que « test-suite/PrintTest/data» contiennent des exemples de code valide et invalide du langage.

1 – Les boucles while

[1]Le site officiel est https://javacc.org/.

Le nom-terminal « WhileStmt » doit être utilisé pour les boucles while. La structure d'une boucle while doit commencer par « while », avoir une condition sous forme d'expression entre parenthèse, et être suivi d'une action (stmt).

Par exemple : « while (true) {} » et « while (count > 0) count = count – 1; » sont des structures valides.

Vous pouvez vous fier sur les tests qui accompagnent le code source pour savoir ce qui est attendu de la grammaire.

2 - les structures conditionnel.
Le non-terminal « IfStmt » doit être utilisé pour les structures conditionnels. La structure doit commencer par « if », avoir une condition sous forme d'expression entre parenthèse, et être suivi d'une action (stmt). Si cette action (stmt) est une série d'action entre accolade, la structure peut être continuer par le token « else » après la fermeture de l'accolade, suivi d'une action (stmt).

Vous pouvez vous fier sur les tests qui accompagnent le code source pour savoir ce qui est attendu de la grammaire.

3 – Expression
Le non-terminal « Expr » est utilisé pour toute les expressions. Une expression est une série d'opération logique ou arithmétique pouvant se résoudre à une valeur. Le langage ne fait pas de différence entre une valeur booléenne et une valeur entière à ce stade-ci.

Les opérateurs d'addition et de soustraction sont déjà présent, ainsi que les opérateurs logiques « et » (&&), « ou » (||) et la gestion des parenthèses.

Ajoutez les opérateurs multiplication (*), division (/), de négation arithmétique (-), et de non logique (!) ainsi que ceux de comparaison (<, >, etc.).

L'ordre de priorité des opérateurs doit être le suivant :

1. Parenthèse
2. Non logique (!)
3. Négation (-)
4. Multiplication (*) et Division (/)
5. Addition (+) et Soustraction (-)
6. Comparaison (<, >, <=, >=, ==, !=)
7. « Et » et « OU » logique (&&, ||)

Vous pouvez ajouter des règles de productions. Pour faciliter la correction, il est encouragé de terminer par « Expr » le nom des non terminaux ajouté (MulExpr, AddExpr, etc.)

Vous pouvez utiliser le test « PrintAllTest » pour voir l'arbre créer par votre grammaire. Il est normal que ce test ne soit jamais validé, puisque l'arbre créer dépend de l'expression tester et des noms de non terminaux que vous aurez choisis.

Lorsque vous ajouté des règles de productions, il sera nécessaire d'adapter les visiteurs pour que les tests fonctionnent à nouveau. Pour ce faire, il suffit d'ajouter la fonction « visit » avec comme paramètre le type associé à la nouvelle règle de productions. IntelliJ permet de générer d'un clic c'est nouvelle fonction en visant de son curseur le début de la classe (là ou une erreur est indiqué) et en utilisant le raccourcis Alt+ Entré pour afficher les options de corrections automatisées.

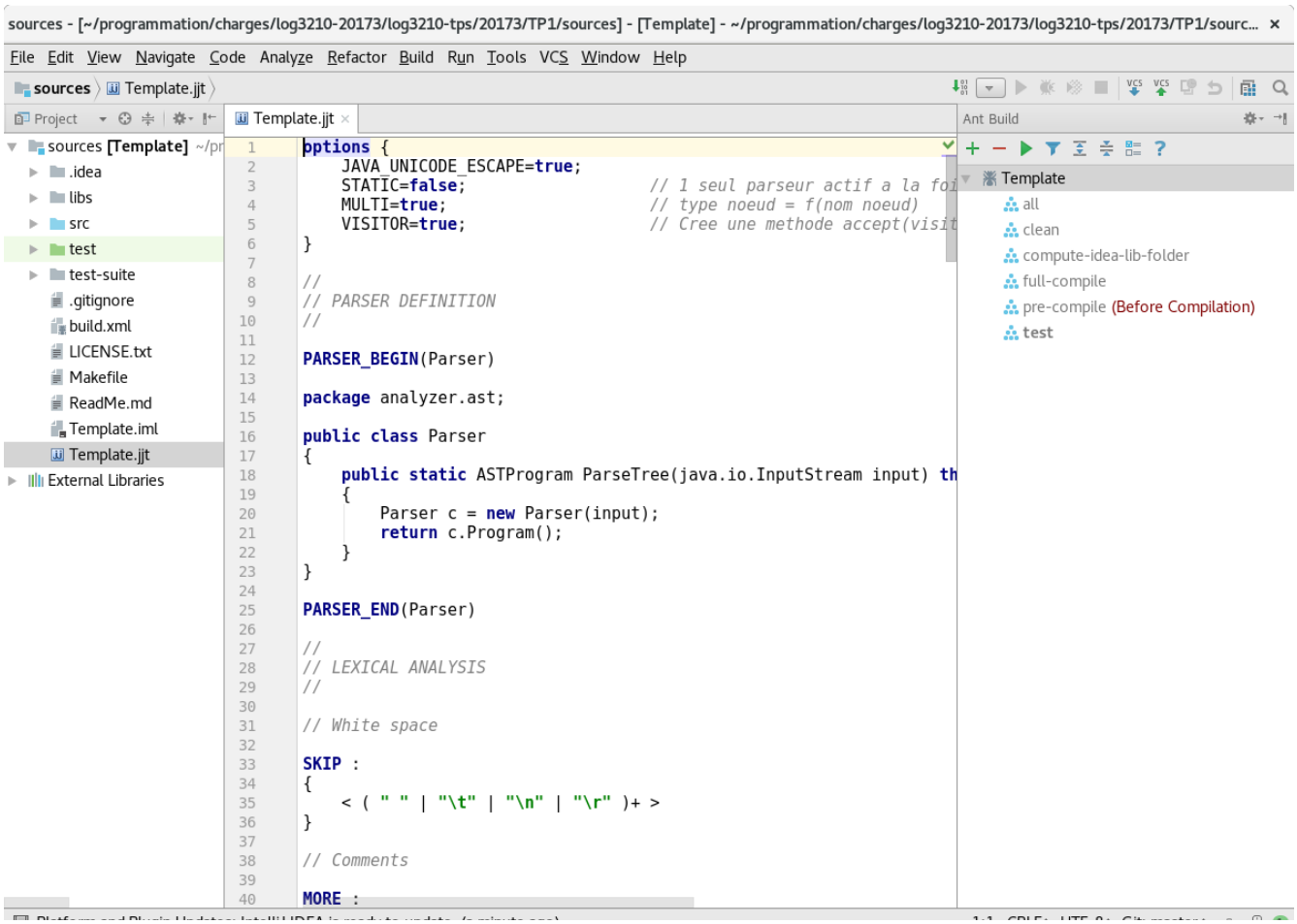
3. Utilisation du cadriciel et de IntelliJ

1. Téléchargez l'archive sur Moodle, puis extrayez-la.
2. Ouvrez IntelliJ.
   À la première ouverture :
   1. N'importez pas les paramètres.
   2. Choisissez votre thème.
   3. Décochez la case pour la création d'une entrée de menu.
   4. Appuyez sur « Skip Remaining and Set Defaults ».
3. Ouvrez le projet (l'archive de Moodle extraite) avec « Open ».
4. Dans la notification qui apparaît au bas de l'écran, suivez les instructions afin d'installer JavaCC, puis redémarrez IntelliJ.
5. Ouvrez le fichier `Template.jjt`.

Vous pouvez désormais apporter vos modifications à la grammaire JavaCC. Pour générer et tester l'analyseur, faites un `all` dans le panneau Ant. (Vous pouvez aussi uniquement générer l'analyseur avec `full-compile` ou exécuter uniquement les tests avec `test`.)

Pour arriver à une fenêtre semblable à celle de la figure suivante :

— Liste des fichiers du projet : appuyez sur `Alt` + `1`
— Panneau Ant : appuyez sur `Ctrl` + `Maj` + `A`, tapez `ant`, puis Entrée



Si vous avez une erreur avec le JDK introuvable, ouvrez la fenêtre *Project Structure* (avec `Ctrl` + `Alt` + `Maj` + `S`), puis sélectionnez New → JDK à côté du champ *Project SDK*. Dans la fenêtre qui apparaît, `java-1.8.0-openjdk` devrait être sélectionné. Appuyez sur OK.

Pour davantage de détails concernant le cadriciel, consulter la page du projet sur GitHub : https://github.com/Nic007/JavaCC-Template.

Le cadriciel du présent TP a été bâti à partir du cadriciel présent sur GitHub. La structure du projet est la même mais le langage demandé est différent.

4. Barème

Le TP est évalué sur 20 points, répartis comme suit :

- 5 points : production du « while »
- 5 points : production du « if »
- 10 points : les productions des expressions.

5. Remise

Remettez sur Moodle une archive nommée log3210-tp1-*matricule1*-*matricule2*.zip avec uniquement le fichier template.jtt et un fichier pdf contenant les réponses aux questions théoriques.

L'échéance pour la remise est le dimanche 27 janvier 2019 à 23 h 55.

Une pénalité de 10 points (50 %) s'appliquera par jour de retard.

Une pénalité de 4 points (20%) s'appliquera si la remise n'est pas conforme aux exigences (nom du fichier de remise, format PDF, fichier template.jtt seulement).