LOG3210 - Élément de langages et compilateur
Intra

Doriane Olewicki – Chargée de cours
Esther Guerrier – Chargée de laboratoire

Hiver 2021

ATTENTION: Cet intra est un devoir individuel !

1 Introduction

Le fameux jour est arrivé!

Votre chargée de laboratoire est arrivée avec le nouveau langage qui révolutionnera le monde de demain: EstheRust. Rapide, clair et élégant, ce langage deviendra encore plus populaire que Python et Java!!! Nous vivons dans un monde où Python règne et les amoureux du Python nous crache dessus et nous humilient à chaque jour...

Aujoud'hui, le 8 mars 2021, nous allons révolutionner la vie des pauvres ingénieurs logiciels. Nous allons mettre un terme à la tyrannie Pythonnienne que Guido van Rossum (créateur de python) a mis en place depuis le 20 février 1991 (première version de python). Nous allons éradiquer cette menace de l'univers jusqu'à notre dernier souffle et préparer un monde meilleur pour les futurs générations!!

Les antis-pythongonistes, rejoignez-moi avec mon arme de destruction massive (EstheRust) pour reprendre le contrôle!!!. Avant d'effacer, de la surface de l'univers, toute trace de python, il faut commencer par se cammoufler des pythongonistes. Pour cela nous allons devoir écrire la grammaire et le visiteur ayant la propriété de convertir du code EstheRust en Python lorsque les pythongonistes observent notre code! Ceci est vital pour pas que les pythongonistes nous remarquent avant le coup d'état!!!!

2 Objectifs

- Écrire la grammaire pour le langage qui nous sauvera des pythongonistes : EstheRust
- Convertir EstheRust en du code python à partir d'un visiteur

3 Travail à faire

Lors de cet intra, vous aurez comme tâche de créer la grammaire d'EsthRust et de convertir ses structures en Python (listée plus bas).

Liste de tâches:

- Créer la grammaire de parsage du langage EstheRust dans le fichier `Langage.jjt`

  N.B.: vous devez créer les Tokens et définir les symboles non-terminaux (fonction) nécessaire à la grammaire. Vous n'êtes **pas** évalués sur la réduction d'arbre. Nous vous avons affiché l'arbre de

parsage pour vous aider à vérifier votre grammaire à cette étape. Nous vous recommandons d'avoir un arbre qui s'affiche pour tous les tests (comme au TP1) **avant** de passer à l'étape suivante.

- Dans le visiteur, définir la fonction visit pour tous les symboles terminaux de votre grammaire. Le code généré dans m_writer doit correspondre à la traduction en Python de votre arbre de parsage.

4 Grammaire EstheRust et traduction en Python

Pour chaque exemple ci-dessous, il y aura la syntaxe en EstheRust à gauche et l'équivalent en Python à droite. Vous aurez à définir les tokens et les fonctions comme vous le voulez et l'objectif est que tous les tests passent. **Attention! L'indentation doit être respectée lors de l'écriture du fichier python. Rappelez-vous que votre fichier Python généré doit être exécutable à la fin!**. Aussi, on considère que Python ne supporte pas les ";" même si dans la vraie vie ce n'est pas le cas. Donc, pas besoin de les écrire dans votre script. Finalement, considéré que la sémantique est bonne et qu'il n'y a pas de vérifications à faire. Votre tâche est seulement de transformer le code en Python.

4.1 Importations

Listing 1: Importations en EstheRust

```
estheRustImportStart
numpy;
panda;
estheRustImportEnd
```

Listing 2: Importations en Python

```
import numpy
import panda
```

4.2 Déclarations

Pour les déclarations, EstheRust supporte plusieurs types: u8, u16, u32, u64, u128, f32, f64, bool et string. On utilise le mot de clé compi pour déclarer des variables.

Listing 3: Déclarations en EstheRust

```
compi var1: u16 = 2;
compi var2: f64 = 4.5;
compi var3: bool = true;
compi var4: string = "sfdsdfsd";
```

Listing 4: Déclarations en Python

```
var1 = 2
var2 = 4.5
var3 = True
var4 = "sfdsdfsd"
```

4.3 Opérations arithmétiques

EstheRust supporte les mêmes opérations mathématiques et booléennes normales (donc comme au TP1 et TP2). Pour cet exercice, vous n'aurez qu'à implémenter:

- Comparaisons ($\&\&, ||, >=, <=, ==, !=$)
- Additions et soustractions ($+, -$)
- Opérations unaires ($-, !$)
- Parenthèses

Par contre, en Python, certaines comparaisons de logiques sont écrites différemment:

- && logique équivaut à **and**

- || logique équivaut à **or**
- ! logique équivaut à **not**

Listing 5: Comparaisons en EstheRust

```
compi comp: bool = true && false;
compi comp2: bool = 3 == 4
compi comp3: bool = !true
```

Listing 6: Comparaisons en Python

```
comp = True and False
comp2 = 3 == 4
comp3 = not True
```

4.4 Fonctions

Les fonctions peuvent avoir de 0 à infini paramètres. On peut aussi appeler des fonctions dans le code. L'appel des fonctions est supporté pour la valeur de retour d'une fonction, dans les conditions et comme déclaration normale. La méthode print n'a pas besoin de supporter le .format pour python. Considérez que les exemples utiliseront la fonction print sans plusieurs paramètres. De plus, pas besoin de transformer le nom des fonctions en pascal case. Finalement, les définitions de fonctions doivent toujours suivre avec au minimum 2 sauts de ligne lors de la traduction en python.

Listing 7: Définition d'une fonction en EstheRust

```
fn greet(name: string) -> bool {
    //code..

    true == false;
}

fn exemple2(name: string) -> () {
    //code..

}

fn emptyFn(name: string) -> () {}
```

Listing 8: Définition d'une fonction en Python

```
def greet(name):
    //code ...

    return True == False

def exemple2(name):
    //code...

def emptyFn(name):
    pass
```

Listing 9: Utilisation d'une fonction en EstheRust

```
fun1();
fun2(true, "ao");
```

Listing 10: Utilisation d'une fonction en Python

```
fun1()
fun2(True, "ao")
```

4.5 Condition If

Pour les if en EstheRust, on ne supporte pas des if sans les curly brackets. Ils doivent TOUJOURS être la.

Listing 11: Condition If en EstheRust

```
if is_divisible_by(n, 15) {
    print("fizzbuzz");
}

if is_divisible_by(n, 15) {}

if true {
    print("fizz");
}

if 3 > 4 {
    print("fizz")
}

if 1 == fonction_booleene() {
    print("buzz");
} else {
    print("bye");
}
```

Listing 12: Condition If en Python

```
if is_divisible_by(n, 15):
    print("fizzbuzz")

if is_divisible_by(n, 15):
    pass

if True:
    print("fizz")

if 3 > 4:
    print("fizz")

if 1 == fonction_booleene():
    print("buzz")
else:
    print("bye")
```

4.6 Boucle While

Listing 13: Boucle while en EstheRust

```
while n < 101 {
    //code...
}

while a_function() {
    //code...
}

while a_function() {}
```

Listing 14: Boucle while en Python

```
while n < 101:
    //code...

while a_function():
    //code...

while a_function():
    pass
```

4.7 Point bonus: Opérateur Coeur

Pour un point bonus (note maximale de 100%), vous pouvez implémenter l'opérateur $*<3*$ en EstheRust qui, en Python, équivaut à print("Compilateurs est le meilleur cours au monde"). Considérer cet opérateur comme un *statement*.

Listing 15: Exemple de l'opérateur Coeur en EstheRust

```
if adore_compi(true) {
    *<3*
    compi_is_life(true);
}
```

Listing 16: Exemple de l'opérateur Coeur traduit en Python

```
if adore_compi(True):
    print("Compilateurs est le
        meilleur cours au monde")
    compi_is_life(True)
```

5 Mode d'emploi pour gérer l'ajout de définitions dans la grammaire et le visiteur

Dans cet intra, nous vous demandons d'écrire une grammaire capable de parser le langage EstheRust ainsi qu'un visiteur. Nous vous conseillons de commencer par faire la partie grammaire au complet avant de vous occuper du visiteur.

Cependant, pour compiler votre code, il est nécessaire que le visiteur `EstheRustConvertorVisitor.java` contienne une fonction `visit` pour tous les symboles non-terminaux (fonction) dans votre grammaire. Ci dessous, vous trouverez comment facilement gérer l'adaptation de votre visiteur lorsque vous rédigez votre grammaire. Pour l'exemple, nous faisons comme-ci on venait d'implémenter le noeud BoolValue déjà fourni pour l'intra et qu'on voulait l'ajouter au projet!

5.1 Création de votre noeud

la première étape consiste en implémentant la syntaxe pour votre fonction/non-terminal/noeud.

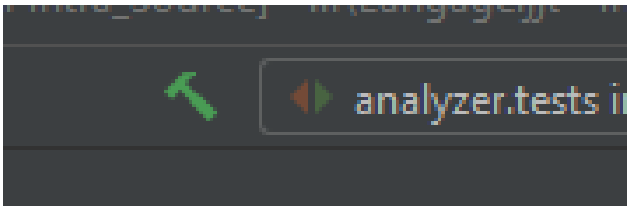
```
void BoolValue() : {Token t;}
{
    t = <BOOL> {jjtThis.setValue(t.image);}
}
```

Dans cet exemple, la valeur lexicale du jeton booléen est stockée dans t, puis l'appel à `jjtThis.setValue` stock cette valeur dans le noeud BoolValue (il est nécessaire de définir la fonction setValue dans la classe BoolValue par la suite, voir les points suivants).

Attention, vous n'aurez pas toujours besoin de stocker une valeur ainsi!

5.2 Génération des fichiers

Lorsque vous avez terminé, pressez le bouton "Marteau" ou "Build" pour laisser JavaCC générer la classe pour vous.



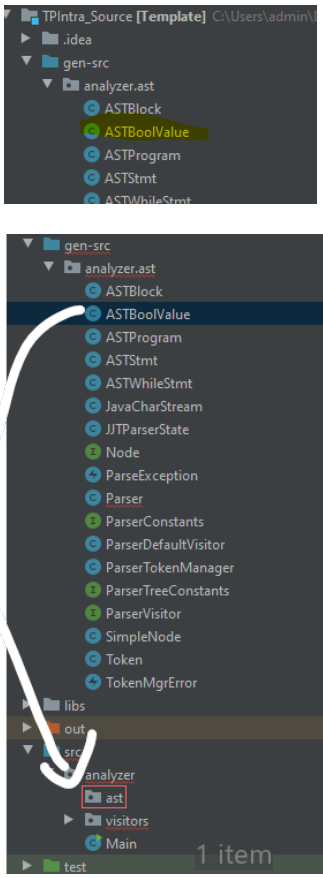
5.3 Vérification de la génération du/des fichiers

Lorsque tu regardes le dossier *gen-src/analyser/ast*, tu devrais apercevoir le/les nouveaux fichiers générés pour les nouvelles fonctions/non-terminaux implémenté dans votre grammaire. Pour nous, on devrait appercevoir ASTBoolValue qui a été géréné pour notre fonction BoolValue.

Si vous avez implémenté du code Java comme dans l'exemple montré ci-dessus au sein de la grammaire (avec le setValue), passez à l'étape 5.4. Sinon, passez directement à l'étape 5.6

5.4 Déplacement du fichier

Pour que JavaCC ne regénère pas la classe que nous allons modifié à la main pour ajouter les méthodes écrites dans le fichier de la grammaire, nous allons déplacer le fichier voulu, ici ASTBoolValue, à *src/analyser/ast*.



5.5 Modification de la classe

Ensuite, on peut ajouter les méthodes que nous voulons à la classe. Pour notre exemple, nous ajouterons l'attribue "value" et ajouterons la méthode setValue nécessaire pour assigner la valeur de l'image du terminal à notre attribut de classe "value". Ensuite, nous implémenterons un getter qui nous permettra d'avoir accès à la valeur au sein de notre visiteur.

5.6 Ajout des méthodes au sein du visiteur

Lorsque vous irez au sein de votre fichier visiteur, il y aura une erreur et c'est normal puisqu'il manque des méthodes "visit" à implémenter pour les noeuds que vous avez implémenté. Pour qu'IntelliJ vous crée toutes les méthodes à implémenter comme par magie, trouver la lumière rouge et cliquez dessus. Il y aura l'option "implement methos" et cliquez dessus.

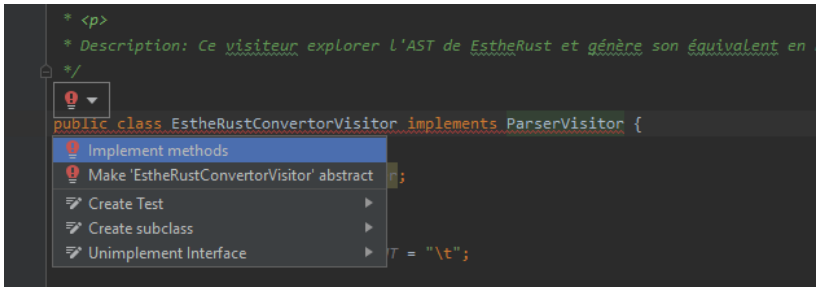
Cela vous ouvrira une fenêtre avec la liste de méthodes qui n'a toujours pas été Override par votre visiteur, pressez "Ok".

Après cela, en refaisant un "build", tout devrait fonctionner et le code devrait compiler.

Vous trouverez aussi une vidéo de démonstration sur Moodle dans la section Intra.

6 Rapport

Nous vous demandons de rédiger un rapport de maximum 2 pages sur votre implémentation: expliquer votre grammaire et le visiteur, ainsi que les objets/fonctions que vous avez créé et utilisé pour répondre à l'intra.



7 Qualité du code et de la grammaire

Concernant votre grammaire, faites attention aux points suivant:

- Grammaire claire et bien organisée;
- Pas de mot entre guillemet, utilisez des Tokens;
- Pas de Lookahead inutile;
- Pas de Tokens inutils (si vous n'avez pas besoin de récupérer de valeur d'un jeton).

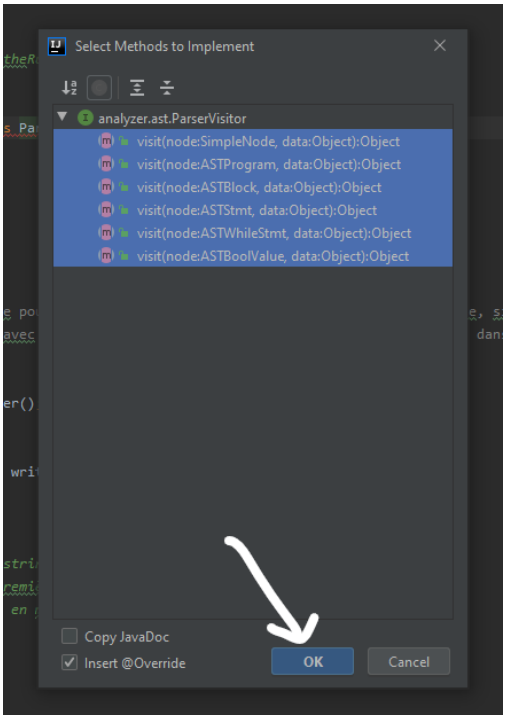
Concernant le visiteur, faites attention aux points suivant:

- Code clair, bien organisé et commenté;
- Ne visitez qu'une fois tout l'arbre (pas de double visite).

8 Barème

La note est répartie Cet intra est évalué sur 20 points, les points étant distribués comme suit :

1. Importations: 1 point;
2. Déclarations: 1 point;



3. Opérations arithmétiques: 3 point;
4. Fonctions: 3 points;
5. Condition If: 2 points.
6. Boucle While: 2 points.
7. Tests surprise: 3 points.
8. Qualité du code et de la grammaire: 2 points.
9. Rapport: 3 points.
10. Opérateur coeur: bonus (+1 point)

Une partie de la note pour chaque point (1 à 7 et 10) est attribuée à la grammaire et l'autre partie à la gestion du visiteur.

Vous n'êtes pas évalué sur l'arbre de parsage (pas besoin de faire de réduction d'arbre comme pour le TP1). Il faut cependant que tous les tests affichent un arbre.

Sachez que tous les scripts python que vous générez doivent être exécutables! Chacun des fichiers que vous générez devront générer la bonne sortie. Les tests se trouvent dans le dossier `test-suite/EstheRustConvertor` où vous pouvez parcourir les dossiers `data` pour les programmes et `expected` pour voir les résultats attendus.

9 Remise

Pour l'intra, vous devez remettre:

- le dossier complet du projet IntelliJ;

- un rapport de max 2 pages au format PDF.

Le tout dans un zip nommée *log3210-intra-votrematricule.zip*.
Le dossier devrait avoir le format:

```
root: log3210-intra-votrematricule.zip
|--- code/
     |--- (contenu du projet)
|--- rapport-votrematricule.pdf
```

Le devoir doit être fait de manière **individuel**. Tout cas de plagiat détecté entrainera automatiquement la note de **0** pour les différentes personnes concernées.

Remettez sur Moodle l'archive nommée *log3210-intra-votrematricule.zip*. L'échéance pour la remise est le **lundi 15 mars 2021 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, manque le rapport, etc.).

Seul les questions de compréhension de l'énoncé seront acceptées pour cet intra. Nous ne vous aiderons pas à résoudre les bugs dans votre code. Si vous avez des questions, veuillez nous contacter sur discord ou, pour une urgence, sur le courriel d'Esther : esther.guerrier@polymtl.ca.