Projet de compilation
Travail à remettre le 18 mai 2020 à minuit dernier délai

Ce que vous avez à faire est juste une modification des fichiers
`StmIF.java`, `StmWHILE.java` `StmFOR.java`, et
`StmSWITCH.java` aux endroits indiqués par `//`**TODO**

version `2020-05-02/17:34:00`

Modalité pour rendre le travail

Déposer dans le Moodle du cours, dans *Rendu projet* :
`https://moodle1.u-bordeaux.fr/mod/assign/view.php?id=204023`

— Un fichier d'archive qui contient tout le code (mais aucune bibliothèque ni aucun fichier compilé)
— Un fichier PDF très court qui contiendra quelques notes à destination du correcteur pour mieux comprendre et évaluer votre dépôt.

1 Introduction

Le langage dont il est question dans ce projet est toujours Léa, qui a été utilisé pour le mini-projet et très légèrement remanié.

Le projet consiste à réaliser un compilateur qui produit du code C à partir du code Léa.

Vous pourrez le tester en lançant la commande ant qui produira les fichiers `data/progr-xxx.c` et `data/progr-xxx.log` à partir des fichiers `data/progr-xxx.lea`. Les fichiers `data/progr-xxx.c` sont les fichiers en langage C que vous pourrez compiler et tester. Les fichiers `data/progr-xxx.log` sont les messages d'erreur que le compilateur Léa produit.

Pour faire ce projet, nous avons réalisé les étapes suivantes :
— Analyse lexicale
— Analyse syntaxique
— Vérification du typage
— Production d'un code abstrait
— Production du code C

Mais malheureusement les instructions `if`, `while`, `for`, et `switch` n'ont pas été implémentées et ne produisent que des messages du type : `--- Manque WHILE...`.

2 Travail à réaliser en groupes de 4 pour 18 mai

Produire dans `generateCode` le code C correspondant aux structures de contrôle `for`, `while`, `if` et `switch`.

Pour vous aider, nous avons commencé à implémenter `if`, mais vous vous rendrez compte qu'il manque la partie `else` qui reste à faire. Complétez-le et inspirez-vous de cela.

L'objectif

Important : Pour cela, il vous est demandé de ne pas utiliser les structures `for`/`while`/`switch` équivalentes en langage C, mais l'utilisation systématique de la commande `goto` précédé éventuellement d'un `if` trivial (i.e. qui ne porte que sur le contenu d'une variable).

Exemple : pour produire l'équivalent de

```
if (XXX) {
  YYY
}
```

Il faut produire ceci :

```
int _t_ = XXX;
if (_t_) goto label_if;
goto label_fin_if;
label_if:
YYY
label_fin_if:
```

En effet, si nous produisons aujourd'hui du code dans un langage qui doit lui-même être compilé c'est seulement pour faciliter l'exercice. Il conviendra à terme (mais pas pour cet exercice) de produire directement du code machine. Ici on se limite donc seulement à des `goto` éventuellement précédés de `if` triviaux car c'est de cela seulement dont on dispose en code machine.

Note : en C, un label ne peut pas être suivi d'une déclaration de variable, seulement d'une instruction. Vous pouvez cependant glisser une instruction vide en ajoutant simplement un bloc vide :

```
monlabel:{}
int _t_ = XXX;
```

3 Éléments fournis

- Fichier pour la compilation `build.xml`
  Il suffit d'utiliser la commande `ant` pour compiler le tout et pour produire le fichier `data/progr-1.output`
- Fichiers exemple `data/progr-1.lea`, ... `data/progr-9.lea`.
- Classe principale
  `fr.ubordeaux.deptinfo.compilation.lea.Main.java`
- Grammaire `CUP`
  `fr.ubordeaux.deptinfo.compilation.lea.parser.Parser.cup`
  `fr.ubordeaux.deptinfo.compilation.lea.parser.Pair.java`
- *Tokenizer* `JFLEX`
  `fr.ubordeaux.deptinfo.compilation.lea.tokenizer.Tokenizer.jflex`

— Environnements
`fr.ubordeaux.deptinfo.compilation.lea.Environment.java`
`fr.ubordeaux.deptinfo.compilation.lea.EnvironmentException.java`
`fr.ubordeaux.deptinfo.compilation.lea.MapEnvironment.java`
Ces classes permettent de conserver l'enregistrement des variables, des types et des constantes.
— Types
`fr.ubordeaux.deptinfo.compilation.lea.Type.java`
`fr.ubordeaux.deptinfo.compilation.lea.TreeType.java`
`fr.ubordeaux.deptinfo.compilation.lea.TypeCode.java`
`fr.ubordeaux.deptinfo.compilation.lea.TypeException.java`
Ces classes permettent de représenter un type simple ou complexe
— Représentation interne du code
  — Interface générale du code. Permet de vérifier le typage et de produire le code C
  `fr.ubordeaux.deptinfo.compilation.lea.CodeInt.java`
  — Les expressions
  `fr.ubordeaux.deptinfo.compilation.lea.Expr.java`
  `fr.ubordeaux.deptinfo.compilation.lea.ExprXXX.java`

  — Les instructions
  `fr.ubordeaux.deptinfo.compilation.lea.Stm.java`
  `fr.ubordeaux.deptinfo.compilation.lea.StmXXX.java`

Vous remarquerez différents outils pour produire du code, tels que `incIndent / decIndent` et `tab()` qui gèrent l'indentation, `NL` pour ajouter des retours à la ligne pour plus de lisibilité.

4 Les fichiers à modifier

— `fr.ubordeaux.deptinfo.compilation.lea.StmIF.java`
— `fr.ubordeaux.deptinfo.compilation.lea.StmWHILE.java`
— `fr.ubordeaux.deptinfo.compilation.lea.StmFOR.java`
— `fr.ubordeaux.deptinfo.compilation.lea.StmSWICH.java`
Conseil : commencez par implémenter les cas les plus simples !