Projet “Compilateur”

Attention Veuillez **lire attentivement l’entier de ce document** qui contient des informations importantes pour le bon déroulement de ce projet et des autres projet du module de Programmation Avancée.

1 Le projet

Ce projet a pour objet la programmation d’un compilateur (ou d’un programme similaire). Il sera réalisé en python en utilisant `PLY`.

Ce projet sera impérativement réalisé par groupes de **deux personnes**[1]. Les groupes devront être **différents** pour chaque projet du module de programmation avancée.

1.1 But

Le but précis du projet est à déterminer par vous-même en fonction de vos intérêts et de vos connaissances préalables. Vous pourrez repartir de la base développée dans les TP’s compilateurs 1–4 ou reprendre sur de nouvelles bases.

Quelques propositions pour ceux qui manquent d’inspiration :

Proposition 1 : compiler vers une machine réelle Définissez un mini-langage de votre choix, ou un sous-ensemble d’un langage existant, et produisez du code assembleur (machine réelle à registres). On peut imaginer par exemple que votre compilateur devra traiter des entrées comme

```
float f(int x) {
  return x+0.1
}
int i = 0;
while (i<10) {
  print i, f(i);
  i++;
}
```

Proposition 2 : compiler vers une machine virtuelle existante Définissez un langage comme dans la proposition 1, et compilez-le, par exemple, vers l’assembleur pour la machine virtuelle java Jasmin : `http://jasmin.sourceforge.net/` (machine virtuelle à pile)

Note Il existe une quantité d’autres machines virtuelles envisageables, à des états de développement ou de documentation variables : Common Language Runtime (.net), Parrot, les machines virtuelles de python, ruby, lua, ...

[1] Dans le cas d’un effectif impair, une (seule) personne travaillera seule ou il y aura un (seul) groupe de 3.

Proposition 2bis : compiler vers une machine virtuelle "maison" Une variante possible de la proposition 2 est d'écrire vous-même une machine virtuelle simple[2] et de générer du *bytecode* pour cette machine.

Proposition 3 : compiler vers un langage de haut niveau Définissez un langage comme dans la proposition 1, mais compilez-le vers du C, ou du Java, ou un autre langage de haut niveau. Notez que pour que le projet soit intéressant, il faudra probablement que le langage source soit significativement différent du langage cible...

Proposition 4 : un interpréteur Définissez comme ci-dessus un langage, mais cette fois-ci dans le but de l'interpréter.

Proposition 5 : un langage de description de documents Définissez un langage de description de documents. Par exemple

```
line (0,0)-(20,0);
text (0,0) "Origine";
for i = 1 to 12 {
  circle (10*i,10*i) radius 5;
}
```

Produisez ensuite un document dans un format de votre choix (Postscript, PDF, SVG, ...)

Propositions 5bis : Un langage de description d'animations Comme en 5, mais plus amusant : plutôt que de produire des images fixes, produisez des animations (en SVG, flash, ...).

Les idées ci-dessus ne constituent que des proposition, mais d'innombrables variantes sont possibles... choisissez quelque chose qui vous motive[3] !

1.2 Technique et contraintes

1.2.1 Technique

- Ce projet sera réalisé en Python et les analyseurs lexical et syntaxique seront basés sur `PLY`.
- L'ensemble du projet est assez libre, mais votre résultat *doit impérativement passer par la construction d'un arbre syntaxique*. Une approche "directe" telle que celle que nous avons pratiqué dans le TP compilateurs 1 *n'est pas suffisante*.
- Le projet devra être conçu pour que les étapes intermédiaires du traitement (lexèmes, AST, éventuellement couture, ...) soient facilement accessibles. Si vous structurez votre projet comme dans les TPs, avec un module par étape que l'on peut importer ou exécuter séparément, il suffit de le documenter. Sinon, vous prévoirez des options en ligne de commande, du genre `--scan-only`, `--parse-only`, etc.

[2] La machine virtuelle pourra éventuellement être réalisée dans un langage compilé pour améliorer l'efficacité. Veuillez en discuter avec le professeur si vous envisagez cette possibilité.

[3] En cas de doute sur la pertinence d'un projet particulier, discutez-en avec le professeur avant de vous y lancer !

1.2.2 Plateforme

Si le projet n'est pas intrinsèquement lié à une plateforme particulière, un effort sera fait pour conserver la portabilité du code (tout est prévu dans python, il suffit de bien l'utiliser !). Dans les faits, sauf raison impérative, votre projet sera corrigé sous Linux. Vous aurez donc intérêt à vous arranger pour qu'il fonctionne sans surprise au moins sur cette plateforme...

2 Démarche

Au cours du projet, vous devrez probablement passer par les étapes suivantes :
- Définition précise du projet (compilateur/interpréteur/..., quels langages source et objet, ...). À ce stade, *informez le professeur* des détails de votre projet
- Écriture et test de l'analyseur lexical
- Écriture et test de l'analyseur syntaxique
- Éventuellement, écriture et test de l'analyseur sémantique
- Écriture et test de la partie arrière
- Intégration, tests du projet complet et écriture de codes sources d'exemple
- Rédaction du mini-rapport

3 Délivrables et délai

À rendre pour le **lundi 11 janvier 2010** au plus tard, par mail à `matthieu.amiguet@he-arc.ch` :
- Tous les fichiers formant la source du projet
- Quelques exemples de codes source acceptés par votre programme et démontrant ses possibilités. Vous pouvez également inclure des codes source erronés et refusés par votre compilateur pour montrer son fonctionnement dans ce cas (surtout si vous avez soigné le traitement des erreurs !)
- Un mini-rapport (au format pdf) décrivant
  - Le but que vous vous êtes fixé
  - Les langages source et objet choisis
  - Les fonctionnalités implémentées
  - Des indications de prise en main (comment utiliser votre compilateur...)
  - Éventuellement, les difficultés rencontrées, les bugs découverts mais non corrigés, un retour d'expérience, ...

4 Évaluation

4.1 Critères

Le projet sera évalué selon les critères suivants :
- Qualité du projet (Originalité, fini, volume de travail, ...)
- Maîtrise technique (Python, `PLY`, techniques de compilation, ...)
- Prise en main (Est-ce que tout marche du premier coup ; qualité et complétude des exemples ; ...)

Le non-respect des consignes (délais, délivrables, ...) ou le plagiat (cf. ci-dessous) pourra amener à une réduction de la note indépendamment des critères ci-dessus.

4.2 Plagiat

Vous développerez vous-même votre code ! la copie d'une (petite) portion de code pré-existant est tolérable si

- elle reste occasionnelle et largement minoritaire ;
- elle est clairement signalée par un commentaire adéquat (avec citation de la source !).

Le code des TP's compilation 1–4 peut être réutilisé sans autre, mais les remarques ci-dessus s'appliquent à tout autre code, y compris les exemples livrés avec `PLY`.

5 Remarques importantes !

- Ne seront prises en compte dans la correction que les fonctionnalités opérationnelles !
  - Évitez les "J'ai *presque* fini d'implémenter le traitement de la fonctionnalité X"
  - Visez donc plutôt un projet modeste, mais fonctionnel, qu'un compilateur C# vers du bytecode java puissant mais où rien ne marche vraiment.
  - Une conception incrémentale de votre compilateur pourra vous aider dans cette voie (commencez par un langage simple et complexifiez-le au fur et à mesure…)
- Évitez d'implémenter des fonctionnalités "internes" inutiles pour le but final (par exemple, coudre l'arbre sans utiliser la couture par la suite).
- Si vous implémentez quelque chose, signalez-le dans le rapport et fournissez un exemple ! il serait dommage que la correction passe côté de la fonctionnalité principale de votre compilateur !
- Nettoyez votre code avant de le rendre !
  - retirez tout code inutilisé ;
  - supprimez les `import` inutiles ;
  - ne laissez pas traîner des blocs de code mis en commentaire pour les désactiver "temporairement"…
- Commentez soigneusement votre code ! (Mais ne confondez pas les commentaires introduits par `#` et les *docstrings* !)

… amusez-vous bien ;–)