TELECOM Nancy 2ème année
Octobre 2018 - Avril 2019

Projet de compilation (Modules PCL1-PCL2)

L'objectif de ce projet est d'écrire un compilateur CLIFF[1] *(*`Compilateur d'un Langage Informatique au Fonctionnement Fonctionnel`*) du langage* `Tiger`*, langage décrit par Andrew Appel (Princeton University) dans son ouvrage sur la théorie de la compilation* `Modern Compiler Implementation`*. Ce compilateur produira en sortie du code assembleur* `microPIUP/ASM`[2]*, code assembleur que vous avez étudié dans le module PFSI de première année.*

1 Réalisation du projet.

Vous travaillerez par groupes de 4 élèves, et en aucun cas seul ou à deux. Si vous êtes amenés à former un trinôme, nous en tiendrons compte lors de l'évaluation de votre projet.

Vous utiliserez l'outil ANTLR, générateur d'analyseur lexical et syntaxique *descendant*, interfacé avec le langage Java pour les étapes d'analyse lexicale et syntaxique. Vous génèrerez ensuite dans un fichier du code assembleur au format *microPIUP/ASM*.
Votre compilateur doit signaler les erreurs lexicales, syntaxiques et sémantiques rencontrées. Lorsqu'une de ces erreurs est rencontrée, elle doit être signalée par un message relativement explicite comprenant, dans la mesure du possible, un numéro de ligne. Votre compilateur peut s'arrêter après chaque erreur syntaxique détectée (pas d'obligation de reprise). En revanche, votre compilateur doit impérativement poursuivre l'analyse après avoir signalé une erreur sémantique.

Vous utiliserez un dépôt Git sur le Gitlab de TELECOM Nancy[3]. Créez votre projet dans votre espace personnel. Il devra être privé et l'identifiant sera de la forme login1 (où login1 est le login du membre chef de projet de votre groupe). Vous ajouterez Suzanne Collin, Sébastien Da Silva et Pierre Monnin en tant que "Master" de votre projet. Votre répertoire devra contenir tous les fichiers sources de votre projet, les dossiers intermédiaire et final (au format PDF) ainsi que le *mode d'emploi* pour utiliser votre compilateur.

Les dépots seront régulièrement consultés par les enseignants chargés de vous évaluer lors des séances de TP et lors de la soutenance finale de votre projet.
En cas de litige sur la participation active de chacun des membres du groupe au projet, le contenu de votre projet sur le dépôt sera examiné. Les notes peuvent être individualisées.

Les modules PCL1 ET PCL2 (qui ne font pas partie du module TRAD1) sont composés de 7 séances de TP. Vous serez évalués en fin du module PCL1 (décembre 2018) où on vous demandera de présenter la grammaire, l'AST et un début de table des symboles, puis en fin du module PCL2 (avril 2019) au cours d'une soutenance lors de laquelle vous présenterez le fonctionnement de votre compilateur dans sa version finale (contrôles sémantiques et génération de code). Vous rendrez aussi en fin des modules PCL1 et PCL2 un petit dossier qui entrera dans l'évaluation de votre projet.

Déroulement et dates à retenir.

PCL1 : séances TP 1 à 4 - Prise en main du logiciel ANTLR, définition complète de la grammaire du langage et de l'AST, début de TDS.

On vous propose une initiation au logiciel ANTLR lors de la première séance. Ensuite, vous définirez la grammaire du langage et la soumettrez à ANTLR afin qu'il génère l'analyseur syntaxique descendant. Bien sûr, l'étape d'analyse lexicale est réalisée parallèlement à l'analyse syntaxique.
Vous aurez testé votre grammaire sur des exemples variés de programmes écrits en TIGER (avec et sans erreurs lexicales et syntaxiques).
Vous réfléchirez ensuite à la construction de l'arbre abstrait que vous mettrez en oeuvre, puis à la table des symboles.

[1] Cet acronyme n'a pas été choisi au hasard...
[2] Le jeu d'instructions et son codage ont été définis par Alexandre Parodi et la syntaxe du langage d'assemblage par Karol Proch.
[3] https://gitlab.telecomnancy.univ-lorraine.fr/

Vous ferez une démonstration de cette étape le mercredi 12 décembre, 14h-18h (un planning des passages sera établi) : vous montrerez la grammaire, l'arbre abstrait et un début de table des symboles (avec visualisation, même sommaire). Vous obtiendrez une note N_1 correspondant à cette première évaluation.

PCL2 : séances TP 6 à 8 - Finalisation de la construction de la TDS, contrôles sémantiques.

Au cours de ces trois séances vous finaliserez la construction de la table des symboles, et vous implémenterez les contrôles sémantiques. A la fin de cette itération (c'est à dire en séance TP8), vous montrerez la table des symboles finale (une visualisation, même sommaire, est indispensable) ainsi que les contrôles sémantiques implémentés. Vous serez évalués et vous obtiendrez une seconde note N_2.

Fin du projet et tests : génération de code assembleur.

Vous continuez votre projet par l'étape de génération de code. Pour cette dernière phase, vous veillerez à générer le code assembleur de manière incrémentale, en commençant par les structures "simples" du langage.

Votre projet sera testé par les enseignants de TP et se fera en votre présence. Il est impératif que vous ayiez prévu des exemples de programmes permettant de tester votre projet et ses limites (ces exemples ne seront pas à écrire le jour de la démonstration) : vous pourrez nous montrer votre "plus beau" programme... Vous obtiendrez alors une note N_3 de démonstration.

Les dossiers.

1. A la fin du module PCL1, vous rendrez un premier rapport présentant :

   - la grammaire du langage,
   - la structure de l'arbre abstrait,
   - la structure de la table des symboles,
   - des *jeux d'essais* illustrant le bon fonctionnement de votre grammaire, l'AST construit ainsi que la TDS.
   - les éléments de gestion de projet relatifs à cette première partie du projet (fiche projet, CR rédigés, Gantt, fiche d'évaluation de la répartition du travail).

2. A la fin du projet (donc du module PCL2) et pour la veille du jour de votre soutenance, vous rendrez un dossier qui complètera le précédent et comprendra *au moins* :

   - la structure de la table des symboles que vous avez définie,
   - les erreurs sémantiques traitées par votre compilateur,
   - les schémas de traduction (du langage proposé vers le langage assembleur) les plus pertinents,
   - des *jeux d'essais* mettant en évidence le bon fonctionnement de votre programme (erreurs correctement traitées, exécutions dans le cas d'un programme correct), et ses limites éventuelles.
   - une fiche d'évaluation de la répartition du travail sur cette seconde période : répartition des tâches au sein de votre binôme, estimation du temps passé sur chaque partie du projet,
   - les divers CR que vous avez rédigés, et le Gantt final.

   Vous remettrez ce dossier dans le casier de votre enseignant de TP.
   Chacun de ces dossiers sera évalué et noté.

Pour finir...

Bien entendu, il est interdit de s'inspirer trop fortement du code d'un autre groupe ; vous pouvez discuter entre-vous sur les structures de données à mettre en place, sur certains points techniques à mettre en oeuvre, etc... mais il est interdit de copier du code source sur vos camarades.

Rappel : les notes peuvent être individualisées.

La fin du projet est fixée au **mardi 23 avril 2019**, date prévue pour les soutenances finales ; lors de cette soutenance, on vous demandera de faire une démonstration de votre projet. Un planning vous sera proposé pour fixer l'ordre de passage des groupes.

Aucun délai supplémentaire ne sera accordé pour la fin du projet. Le temps restant sur les mois d'avril et de mai est réservé à la finalisation de vos autres projets et du PIDR.

2 Présentation du langage.

Il est inutile de retranscrire le manuel de référence du langage édité par les auteurs de Tiger. Une version de ce manuel de référence est fournie avec le sujet et vous servira pour construire la grammaire et implémenter les contrôles sémantiques.

Quelques simplifications sont proposées :

- les commentaires ne seront pas imbriqués,

- les chaînes de caractères sont composées d'une suite non vide de caractères imprimables. Elles s'écrivent entre des guillemets (caractère ").

- les entrées/sorties seront limitées aux fonctions **read** et **print**. La fonction prédéfinie **read** lit un entier sur l'entrée standard et la fonction **print** écrit sur une ligne de la sortie standard soit un entier, soit une chaîne de caractères. Les termes **read** et **print** sont des mots-clés du langage.

3 Génération de code.

Le code généré devra être en langage d'assemblage *microPIUP/ASM* écrit dans un fichier texte au format Linux d'extension `.src` , en utilisant un sous-ensemble des instructions de la machine.
Le fichier généré devra être assemblé à l'aide de l'assembleur qui générera un fichier de code machine d'extension `*.iup`.
Ce dernier sera exécuté à l'aide du simulateur du processeur `APR`[4].
Ces deux outils (assembleur et simulateur) fonctionnent sur toute machine Windows ou Linux disposant d'un runtime java. Ils sont inclus dans le fichier (archive java) `microPIUP.jar` disponible sur le serveur neptune dans le dossier `/home/depot/PFSI`.

Ce fichier supporte les commandes suivantes, en supposant que le fichier `microPIUP.jar` soit dans le dossier courant.

1. Assembler un fichier `projet.src` dans le fichier de code machine `projet.iup` :
   `java -jar microPIUP.jar -ass projet.src`
2. Exécuter en batch le fichier de code machine `projet.iup` :
   `java -jar microPIUP.jar -batch projet.iup`
3. Lancer le simulateur sur interface graphique :
   `java -jar microPIUP.jar -sim`

[4] Advanced Pedagogic RISC développé par Alexandre Parodi qui comporte toutes les instructions et modes d'adressage pour permettre l'enseignement général de l'assembleur, mais dont un sous-ensemble forme une machine RISC facilitant l'implémentation matérielle sur une puce et (on l'espère) l'écriture d'un compilateur.