TELECOM Nancy 2ème année
Janvier 2016 - Mai 2016

Projet de compilation (Module PCL).

L'objectif de ce projet est d'écrire un compilateur SCOOL (`Small Compiler Of Original Language`) *du langage* `Tiger`*, langage décrit par Andrew Appel (Princeton University).* [1]
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 dépot (SVN ou `git`) sur la forge de l'école : (`https://forge.telecomnancy.univ-lorraine.fr`). Créez votre projet comme un sous-projet de COMPILATION_2A. (dans Projets Divers). Votre projet doit être privé, l'identifiant sera de la forme `login1` (où `login1` est le login du membre chef de projet du groupe). Vous ajouterez Sébastien Da Silva, Pierre Monnin et Suzanne Collin aux membres développeurs de votre projet. Votre répertoire devra contenir tous les sources de votre projet, le dossier final (au format pdf) ainsi que le *mode d'emploi* pour utiliser votre compilateur.
Les dépots seront consultés par les enseignants chargés de vous évaluer lors de la séance de soutenance 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 la forge sera examiné. Les notes peuvent être individualisées.

Le module PROJET DE COMPILATION - PCL (qui ne fait pas partie du module TRAD1) est composé de 7 séances de TP : vous serez évalué en fin de projet (23 mai 2016) lors d'une soutenance au cours de laquelle vous présenterez le fonctionnement de votre compilateur, mais également au cours des différentes séances de TP qui composent le module. Vous rendrez aussi en fin de projet un dossier qui entrera dans l'évaluation de votre projet.

Déroulement et dates à retenir.

Séances TP 1 à 4 : prise en main du logiciel ANTLR, définition complète de la grammaire du langage et construction de l'arbre abstrait.
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 SCOOL (avec et sans erreur lexicales et syntaxiques).
Une fois la grammaire définie, il vous est demandé d'y intégrer les fonctions de construction de l'arbre abstrait.
Vous ferez impérativement une démonstration de cette étape lors de la séance TP 4 au plus tard. Vous obtiendrez une note N_1 correspondant à cette première évaluation.

1. Ce langage est décrit dans son ouvrage `Modern Compiler Implementation - Cambridge University Press`.
2. Le jeu d'instructions et son codage ont été définis par Alexandre Parodi et la syntaxe du langage d'assemblage par Karol Proch.

Séances TP 5 et 6 : construction de la table des symboles et mise en place des contrôles sémantiques.
Au cours de ces deux séances vous mettrez en place la table des symboles ainsi que les contrôles sémantiques. A la fin de cette itération, vous serez évalués sur ces deux points ; vous montrerez la structure de table des symboles sur des exemples de programmes de test (une visualisation, même sommaire, est indispensable) et vous aurez créé des exemples de programmes contenant des erreurs sémantiques que votre compilateur est capable de détecter. Vous obtiendrez une seconde note N_2 en séance TP 6.

Séance TP 7 et fin du projet.
Vous continuez votre projet en mettant en oeuvre la génération de code. Pour cette dernière étape, vous veillerez à générer le code assembleur de manière incrémentale, en commençant par les structures "simples" du langage. Ainsi, vous respecterez l'ordre suivant pour la génération de code : variables et expressions simples (affectations et expressions arithmétiques), conditionnelles, itérations, fonctions, tableaux, structures.

Les tests finaux.

La fin du projet est fixée au **lundi 23 mai 2016**, 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.
Votre projet sera testé par les enseignants de TP 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 et soutenance.

Le dossier.

A la fin du projet et pour la veille du jour de de votre soutenance, vous rendrez un dossier (qui fournira une note N_4) comportant une présentation de votre réalisation. Ce dossier comprendra *au moins* :

- la grammaire du langage,
- la structure de l'arbre abstrait et de la table des symboles que vous avez définis,
- les erreurs 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 : répartition des tâches au sein de votre binôme, estimation du temps passé sur chaque partie du projet.

Vous remettrez ce dossier dans le casier de votre enseignant de TP.

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. Et bien sûr, pour toute idée trouvée sur la toile, vous en citerez les sources.

La note finale (sur 20) de votre projet prend en compte les 4 notes attribuées lors des différentes évaluations. *Rappel : les notes peuvent être individualisées.*

Aucun délai supplémentaire ne sera accordé pour la fin du projet : les notes doivent être harmonisées et attribuées pour le jury du 26 mai.

2 Présentation du langage.

Inutile de retranscrire le manuel de référence du langage édité par les auteurs de `Tiger` ! Une version de ce manuel de référence est donc fournie avec le sujet et vous servira pour construire la grammaire et développer les contrôles sémantiques.

Ce manuel est amendé de quelques simplifications :

Section "Lexical Aspects" :

- 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 “).

Section "Standard Library" :

Cette section sera très simplifiée, et comportera uniquement les entrée/sortie suivantes : `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`[3].
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 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`

3. 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.