TELECOM Nancy 2ème année
Octobre 2021 - Janvier 2022 - Avril 2022

Projet de compilation : Modules PCL1 et PCL2

L'objectif de ce projet est d'écrire un compilateur d'un langage dit de "haut niveau", en développant toutes les étapes depuis l'analyse lexicale jusqu'à la production de code assembleur. L'an passé, le projet de compilation de l'Ecole polytechnique consistait à écrire un compilateur pour un fragment du langage C, entièrement compatible avec C. Nous reprenons pour le projet PCL les spécifications de leur langage Mini-C. *Elles conviennent tout à fait à nos objectifs, vous faire re-travailler sur ce langage... Ce "sous-langage" C, dénommé* CIR-C *pour votre projet*[1] *vous permettra d'utiliser un compilateur C existant comme référence (tel* gcc *) pour tester vos programmes.*

Le sujet ci-dessous décrit le langage CIR-C ainsi que le travail à réaliser.

1 Réalisation du projet

Cette année, les modules PCL1 et PCL2 ne sont plus des modules du tronc commun. La partie PCL1 (d'octobre à mi-janvier) concernera tous les élèves : il s'agit d'écrire la grammaire du langage, puis de mettre en oeuvre les analyses lexicale, syntaxique et sémantique. La partie PCL2 (de février à fin avril) concernera uniquement la génération de code assembleur. Seuls les élèves des approfondissements IL, ISS, LE sont concernés par ce module.

Pour ce projet, les élèves des approfondissements IAMD, SIE formeront des groupes de 3, tandis que les élèves des autres approfondissements travailleront par groupes de 4.
Les 5 élèves en mobilité au semestre 8 formeront un groupe, et ce quel que soit leur approfondissement, puisqu'ils ne sont concernés que par la partie PCL1.

Vous utiliserez l'outil ANTLR-4, générateur d'analyseurs lexical et syntaxique *descendant*, interfacé avec le langage Java pour les étapes d'analyse lexicale et syntaxique. Vous génèrerez ensuite le code assembleur au format ARM. Votre compilateur doit signaler les erreurs lexicales, syntaxiques et sémantiques rencontrées. Lorsqu'une de ces erreurs lexicales ou sémantique est rencontrée, elle doit être signalée par un message 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. Vous devrez créer 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 Alexandre Bourbeillon, Suzanne Collin et Sébastien Da Silva en tant que "Master" de votre projet. Votre répertoire devra contenir tous les fichiers sources de votre projet, les dossiers intermédiaires et finals (au format PDF) ainsi qu'un *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.

2 Première partie : module PCL1

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

On vous propose une initiation au logiciel ANTLR lors des deux premières séances de TP. Vous définirez ensuite 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.

[1] Un "clin d'oeil à la magicienne grecque aux terribles pouvoirs de transformations, comme le sera votre compilateur" - Et aussi, "CIR Is Reduced C" - Christophe Bouthier.

Vous aurez testé votre grammaire sur des exemples variés de programmes écrits en langage CIR-C, avec et sans erreurs lexicales et syntaxiques.
Vous réfléchirez ensuite à la construction de l'arbre abstrait et de la table des symboles. Vous implémenterez ensuite tous les contrôles sémantiques liés à ce langage.

La soutenance finale de cette partie PCL1 est fixée au 11 et 12 janvier 2022.

Vous montrerez lors de cette soutenance l'arbre abstrait, la table des symboles (une visualisation, même sommaire, est indispensable) ainsi que l'ensemble des contrôles sémantiques implémentés.

Il est impératif que vous ayez prévu pour la soutenance des exemples de programmes permettant de tester votre projet. Ces exemples ne seront pas à écrire le jour de la démonstration...

Vous rendrez au cours de ce module des **rapports d'activité** (max. 4 pages) permettant de faire le point sur l'avancement de votre travail, les difficultés rencontrées, et présentant les futures étapes à réaliser. Vous pouvez inclure dans ces documents la grammaire du langage, la structure de l'arbre abstrait et de la structure de la table des symboles, les jeux d'essais, etc. N'oubliez pas les éléments de gestion de projet (répartition du travail et des tâches au sein du groupe par exemple).

Vous remettrez ces rapports d'activité à votre enseignant de TP les semaines 46, 49 et le lundi 10 janvier à 12h dans son casier.

3 Seconde partie : module PCL2

Génération de code assembleur ARM

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.

Le code généré devra être en langage d'assemblage ARM, langage étudié dans le module *ASM* de première année.

Des séances de TP sont prévues pour cette seconde partie PCL2. Nous reviendrons vers vous pour les consignes

A la fin du projet, donc à la fin du module PCL2, et pour le **vendredi 22 avril 2022 - 18h**, vous rendrez un dossier qui complètera les précédents rapports d'activités et comprendra *au moins* :

- les schémas de traduction du langage proposé vers le langage assembleur
- des *jeux d'essais* mettant en évidence le bon fonctionnement de votre compilateur, et ses limites éventuelles.
- une partie *gestion de projet*, à savoir une fiche d'évaluation de la répartition du travail sur cette seconde période avec la répartition des tâches au sein de votre binôme, l'estimation du temps passé sur chaque partie du projet, et le Gantt final.
- les divers CR de réunion que vous avez rédigés.

Vous remettrez ce dossier dans le casier de votre enseignant de TP le lundi 25 avril 2022.

La fin du projet est fixée au mardi 26 avril 2022, date prévue pour les soutenances finales.

Lors de cette soutenance, on vous demandera de faire une démonstration de votre compilateur. Un planning vous sera proposé pour fixer l'ordre de passage des groupes.
Il est impératif que vous ayez 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 nous montrerez lors de votre soutenance votre "plus beau" programme...

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.

Concernant l'évaluation de la génération de code, on vous fournira 4 "niveaux" de programmes écrits dans le langage du projet, ce qui vous permettra de vous situer dans cette phase de génération de code. Le niveau 2 est

généralement celui requis pour obtenir une note de 10/20. Ceci n'est qu'une indication, car la note finale dépend aussi des évaluations intermédiaires et du dossier rendu.
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. Si cela devait se produire, nous saurons en tenir compte dans nos évaluations.

4 Présentation du langage

La section qui suit décrit la spécification du langage CIR-C. La sémantique associée est celle du langage C.

4.1 Conventions lexicales

Les espaces, tabulation et retour-chariots constituent les "blancs" dans le texte d'un programme. Les commentaires pourront prendre les 2 formes suivantes :

- ils débutent par `/*` et se terminent par `*/`. Ils ne peuvent pas être imbriqués.
- ils débutent par `//` et s'étendent jusqu'à la fin de la ligne.

Les *identificateurs* d'un programme commencent par une lettre minuscule ou majuscule et peuvent ensuite être constitués d'une répétition de lettres (minuscule ou majuscule) ou de chiffres de 0 à 9 ou du caractère `_`.

Les mots clés du langage sont les suivants :

```
int if else while struct return sizeof
```

Les constantes littérales devront respecter les règles suivantes :

```
<chiffre>    ::= 0-9
<entier>     ::= 0
                 | 1-9 <chiffre>*
                 | '<caractère>'
<caractère>  ::=  tout caractère du code ASCII compris entre 32 et 126 autre que \ ' "
                 | \\
                 | \'
                 | \"
```

4.2 Syntaxe

Les fichiers sources que vous écrirez respecteront la grammaire donnée figure 1, page suivante. Dans cette grammaire, l'axiome est le non-terminal `<fichier>`.

Les associativités et précédences des opérateurs sont données dans la table ci-dessous, de la plus faible à la plus forte précédence.

opérateur	associativité	précédence
=	à droite	plus faible
||	à gauche	
&&	à gauche	
== !=	à gauche	
< <= > >=	à gauche	
+ −	à gauche	
* /	à gauche	
! − (unaire)	à gauche	
−>	à gauche	plus forte

$$
\begin{array}{lcl}
\langle fichier \rangle & ::= & \langle decl \rangle^* \ \texttt{EOF} \\
\langle decl \rangle & ::= & \langle decl\_typ \rangle \mid \langle decl\_fct \rangle \\
\langle decl\_vars \rangle & ::= & \texttt{int}\ \langle ident \rangle^+_{,}\ \texttt{;} \\
 & \mid & \texttt{struct}\ \langle ident \rangle\ (\texttt{*}\ \langle ident \rangle)^+_{,}\ \texttt{;} \\
\langle decl\_typ \rangle & ::= & \texttt{struct}\ \langle ident \rangle\ \texttt{\{}\ \langle decl\_vars \rangle^*\ \texttt{\}}\ \texttt{;} \\
\langle decl\_fct \rangle & ::= & \texttt{int}\ \langle ident \rangle\ \texttt{(}\ \langle param \rangle^*_{,}\ \texttt{)}\ \langle bloc \rangle \\
 & \mid & \texttt{struct}\ \langle ident \rangle\ \texttt{*}\ \langle ident \rangle\ \texttt{(}\ \langle param \rangle^*_{,}\ \texttt{)}\ \langle bloc \rangle \\
\langle param \rangle & ::= & \texttt{int}\ \langle ident \rangle \mid \texttt{struct}\ \langle ident \rangle\ \texttt{*}\ \langle ident \rangle \\
\langle expr \rangle & ::= & \langle entier \rangle \\
 & \mid & \langle ident \rangle \\
 & \mid & \langle expr \rangle\ \texttt{->}\ \langle ident \rangle \\
 & \mid & \langle ident \rangle\ \texttt{(}\ \langle expr \rangle^*_{,}\ \texttt{)} \\
 & \mid & \texttt{!}\ \langle expr \rangle \mid \texttt{-}\ \langle expr \rangle \\
 & \mid & \langle expr \rangle\ \langle opérateur \rangle\ \langle expr \rangle \\
 & \mid & \texttt{sizeof (struct}\ \langle ident \rangle\ \texttt{)} \\
 & \mid & \texttt{(}\ \langle expr \rangle\ \texttt{)} \\
\langle opérateur \rangle & ::= & \texttt{=} \mid \texttt{==} \mid \texttt{!=} \mid \texttt{<} \mid \texttt{<=} \mid \texttt{>} \mid \texttt{>=} \mid \texttt{+} \mid \texttt{-} \mid \texttt{*} \mid \texttt{/} \mid \texttt{\&\&} \mid \texttt{||} \\
\langle instruction \rangle & ::= & \texttt{;} \\
 & \mid & \langle expr \rangle\ \texttt{;} \\
 & \mid & \texttt{if (}\ \langle expr \rangle\ \texttt{)}\ \langle instruction \rangle \\
 & \mid & \texttt{if (}\ \langle expr \rangle\ \texttt{)}\ \langle instruction \rangle\ \texttt{else}\ \langle instruction \rangle \\
 & \mid & \texttt{while (}\ \langle expr \rangle\ \texttt{)}\ \langle instruction \rangle \\
 & \mid & \langle bloc \rangle \\
 & \mid & \texttt{return}\ \langle expr \rangle\ \texttt{;} \\
\langle bloc \rangle & ::= & \texttt{\{}\ \langle decl\_vars \rangle^*\ \langle instruction \rangle^*\ \texttt{\}}
\end{array}
$$

FIGURE 1 – Grammaire des fichiers C.

4.3 Sémantique - Typage statique

Une fois l'analyse syntaxique de votre programme source effectuée avec succès, il faut vérifier qu'il est conforme à la sémantique du langage. Il s'agit alors de vérifier le typage des expressions, des instructions, vérifier l'unicité des déclarations de variables, de fonctions, de paramètres et de champs dans les fichiers, blocs, structures. Vous consulterez à ce sujet la documentation du langage C : la sémantique du langage de votre projet est la même que celle du langage C.

Messages d'erreur. Vous pouvez vous inspirer des messages d'erreur fournis par un compilateur C existant.

Fonctions prédéfinies. Les fonctions suivantes sont supposées prédéfinies et devront être connues à l'analyse sémantique :

```
void print(int n);
void *malloc(int n);
```

Point d'entrée. Vous vérifierez la présence d'une fonction `main()` avec le profil suivant : `int main();`

Limites par rapport au langage C. Tout programme correct écrit en CIR-C est compatible C, c'est à dire que c'est aussi un programme C correct. Certaines limitations sont cependant à noter :

- il n'y a pas d'initialisation des variables par défaut lors des déclarations,
- il n'y a pas d'arithmétique des pointeurs,
- il y a évidemment moins de constructions syntaxiques (donc de mots clés) que dans le langage C standard.