TELECOM Nancy 1ère année - Apprentissage
Mars 2014 - Juin 2014

Projet de compilation.

L'objectif de ce projet est d'écrire un compilateur du langage `LEAC` *(Langage Elémentaire Algorithmique pour la Compilation), dont les constructions sont proches de celles du langage Pascal et du langage C que vous connaissez bien. Ce compilateur produira en sortie du code écrit en langage C.*

1 Réalisation du projet.

Vous travaillerez par groupes de 3 ou 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 écrit en langage C. 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 doit également essayer de poursuivre l'analyse après avoir signalé une erreur sémantique.

Vous devrez utiliser un dépot SVN sur la forge de TELECOM Nancy (`https://forge.esial.uhp-nancy.fr`). Créez votre projet comme un sous-projet de APPRENTISSAGE_1A COMPILATION. 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 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.

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.

Le module PROJET DE COMPILATION (qui ne fait pas partie du module COMPILATION) est composé de 7 séances de TP : vous serez évalué en fin de projet (courant juin 2014) 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 1 à 4 : prise en main du logiciel ANTLR et définition complète de la grammaire du langage.

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 Leac (avec et sans erreur lexicales et syntaxiques).

Vous ferez impérativement une démonstration de cette étape lors de la séance de la séance 4 au plus tard. Vous obtiendrez une note N_1 correspondant à cette première évaluation. Il n'est pas interdit de prendre de l'avance et de commencer la construction de l'arbre abstrait au cours de ces 4 séances...

Séances 5 et 6 : construction de l'arbre abstrait et de la table des symboles.

Au cours de ces deux séances vous réfléchirez à la construction de l'arbre abstrait et de la table des symboles. A la fin de cette itération, vous montrerez ces deux structures sur des exemples de programmes de test (une visualisation, même sommaire, de ces structures est indispensable). Vous serez évalués et vous obtiendrez une seconde note N_2 en séance 6.

Séance 7 et fin du projet.

Vous continuez votre projet en mettant en place la phase d'analyse sémantique (si ce n'est pas déjà fait) et la génération de code. Pour cette dernière étape, vous veillerez à générer le code C de manière incrémentale, en commençant par les structures "simples" du langage.

Les tests.

Votre projet sera testé par l'enseignant 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 obtiendrez alors une note N_3 de démonstration.

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 sur vos camarades.

La note finale (sur 20) de votre projet prend en compte les 4 notes attribuées lors des différentes évaluations.

La fin du projet est fixée au **11 juin 2014**, date à laquelle on vous demandera de faire une démonstration de votre projet. Un planning sera proposé pour fixer l'ordre de passage des groupes de projet.

Aucun délai supplémentaire ne pourra être accordé pour la fin du projet.

2 Présentation du langage.

Aspects lexicaux.

Identificateurs : Un *identificateur* est composé des lettres de l'alphabet, majuscules ou minuscules, et des chiffres de 0 à 9, à l'exception de tout autre caractère.
Un identificateur commence obligatoirement par une lettre. Les majuscules et minuscules sont différenciées. Dans la suite du sujet, le symbole `IDF` sera utilisé comme synonyme d'identificateur.
Commentaires : Un commentaire peut apparaître n'importe où dans le texte source : les commentaires commencent par /* et se terminent par */. Ils ne sont pas imbriqués.

Le langage - aspects syntaxiques.

On donne ci-dessous la grammaire complète du langage `LEAC`. Dans cette grammaire, les non-terminaux sont en lettres minuscules et les terminaux en lettres majuscules. Les autres symboles, tels **`( ) + -`** etc sont aussi des symboles terminaux et sont écrits en caractères gras. Les mots-clés seront en minuscules et également en gras. Le terminal CSTE représente les constantes entières, booléennes (`true` ou `false`) ou chaînes de caractères (écrites entre guillemets). Le terminal IDF représente les identificateurs, cf. paragraphe précédent. Le type `void` sert uniquement à spécifier le type de la valeur retournée par une fonction dont le seul but est de causer un effet de bord (cette fonction est donc une procédure).
Le mot clé `var` permet de déclarer des variables locales à un bloc.

Le symbole | désigne l'alternative dans la grammaire et $\wedge$ le mot vide.

program	→	**program** IDF vardeclist fundeclist instr
vardeclist	→	∧ | varsuitdecl vardeclist
varsuitdecl	→	**var** identlist **:** typename **;**
identlist	→	IDF | IDF **,** identlist
typename	→	**void** | **bool** | **int**
fundeclist	→	∧ | fundecl fundeclist
fundecl	→	**function** IDF **(** arglist **) :** typename vardeclist instr
arglist	→	∧ | arg | arg **,** arglist
arg	→	IDF **:** typename | **ref** IDF **:** typename
instr	→	**if** expr **then** instr **else** instr
		| **while** expr **do** instr | lvalue = expr | **return** expr | **return**
		| IDF **(** exprlist **)** | IDF **( )** | **{** sequence **}** | **{ }**
		| **read** lvalue | **write** lvalue | **write** CSTE
sequence	→	instr**;** sequence | instr**;**
lvalue	→	IDF
exprlist	→	expr | expr **,** exprlist
expr	→	CSTE | **(** expr **)** | expr opb expr | opun expr
		| IDF **(** exprlist **)** | IDF **( )** | IDF
opb	→	+ | - | * | / | < | <= | > | >= | == | != | and | or
opun	→	- | not

Le langage - aspects sémantiques.

Il n'y aura pas de bloc sans nom dans le langage du projet. Les fonctions seront systématiquement déclarées avant leur utilisation (pas de "prototype" comme en C).
Le mot-clé `ref` dans une fonction désigne un mode de passage des paramètres par adresse. En l'absence de ce mot-clé, le mode de passage des paramètres est le mode par valeur.
Les expressions seront évaluées de gauche à droite avec les priorités habituelles des opérateurs arithmétiques et logiques.

Portée des déclarations et visibilité des variables.

Des variables dites globales pourront être déclarées en début du programme : la portée de la déclaration de ces variables globales est tout le fichier et elles seront visibles dans toute la partie du fichier située après leur déclaration. Les variables définies dans un bloc sont visibles seulement dans ce bloc. Les paramètres d'une fonction sont visibles uniquement dans cette fonction.

Il n'y a qu'un seul espace des noms.

Entrées-Sorties.

Pour les entrées-sorties, on utilisera les opérations `read` et `write` qui réalisent respectivement la lecture à partir d'une entrée au clavier et l'écriture sur la sortie standard qu'est l'écran.

Exemples de programmes.

Le programme suivant est un exemple de programme simple écrit en `LEAC`.

```
/* Un  exemple de programme écrit dans un super langage */
program essai
   var i, j, maximum: int;
   var Tval : array[-3..3, 0..5] of int;

   function maxTAB (t:  array[-3..3, 0..5] of int) : int
      var i, j, max: int;
      { i = -3;
        j = 0;
        max = t[-3, 0];
        while i <= 3 do
          {while j <= 5 do
             { if t[i, j] > max then max = t[i, j];
               j = j + 1
             }
             i = i + 1
          }
       return max
      }

    function theEnd () : void
       { write "that's all !" }

 { /* début du programme principal */
   i = -3;
   j = 0;
   maximum = t[-3, 0];
   while i <= 3 do
     {while j <= 5 do
        { read Tval[i, j];
          j = j + 1
        }
        i = i + 1
     }
   maximum = maxTAB(Tval);
   write maximum;
   theEnd()
}
```

3 Génération de code.

Le code généré devra être en langage C, écrit dans un fichier texte d'extension `.src`. Ce code produit sera ensuite compilé à l'aide du compilateur *gcc* : il produira ainsi le code cible du fichier source.