Université Bordeaux I **Compilation – INF466**
Master Informatique **Université Bordeaux 1 – 2013-2014**

Projet de Compilation
à remettre le 10 décembre 2013

Modalités pratiques

Ce projet est à réaliser en binôme ou trinôme, pour le 10 décembre 2013 au plus tard.

Vous devrez envoyer un courrier électronique à l'enseignant de votre groupe de TD en mettant en copie l'autre enseignant : `frederique.carrere@labri.fr` ET `lionel.clement@labri.fr`

- L'objet est "DM Compilation"
- Signé des prénoms et noms de tous les participants du projet
- Contenant les trois fichiers attachés suivants :

1. `dm.tar.bz2`, une archive `tar` compressée avec bz2, en excluant tout autre format d'archivage. Cette archive contiendra toutes les sources et toutes les données permettant la compilation et l'exécution du programme complet par simple appel à la commande `ant` dans un terminal.
2. `dm.pdf`, une documentation rédigée de votre travail, sous la forme d'un texte en format `.pdf` en excluant tout autre format. Ce texte contiendra :
   (a) Une spécification du langage choisi illustrée d'exemples
   (b) Une explication des choix d'implémentation retenus
   (c) Les problèmes rencontrés et les solutions que vous y avez apportées
3. `input.txt`, un programme qui peut être donné en *input* de votre application

Le but de ce projet est d'écrire un compilateur pour un langage procédural déclaratif. Le compilateur doit produire du code intermédiaire à trois adresses. Les parties notées avec $^{(*)}$, $^{(**)}$, et $^{(***)}$ sont plus compliquées et correspondent à trois alternatives. Vous développerez l'une des trois seulement en le précisant dans la documentation.

Le langage procédural source sera conçu par vous-même. Il doit permettre d'écrire :

- Les commentaires sur une ou plusieurs lignes
- Les déclarations de variables typées statiquement
- Les types simples :
  - Entier signé sur 1 et 2 octets (-128...127, -32768...32767)
  - Entier non signé sur 1 et 2 octets (0...255, 0...65535)
  - Booléen 1 octet
  - Caractère 4 octets (utf-32)
  - Réel sur 4 octets ($1.5\ 10^{-45}$...$3.4\ 10^{38}$)
  - Enumération sur 1 octet. On ne peut pas énumérer plus de 255 éléments distincts.
- Les types complexes :
  - Intervalles sur des entiers ou des énumérations
  - Chaines de caractères sur n caractères ($1 \leq n \leq 65535$)
  - Tableaux à une dimension $Array(t)$ où t un type quelconque
  - Tableaux à plusieurs dimensions$^{(*)}$ $Array(i_1, i_2, \ldots i_k, t)$ où i_i est un intervalle et t un type quelconque
  - Enregistrements$^{(**)}$
  - Pointeurs sur des éléments d'un type donné

- Les expressions arithmétiques et logiques
- Les instructions :
  - Les appels de procédures$^{(***)}$
  - Les affectations
  - Les tests
  - Les boucles
  - Les blocs (suite d'instructions précédée de déclarations locales à ces instructions)
- Les déclarations de procédures$^{(***)}$qui peuvent contenir des appels récursifs mais qui ne peuvent pas contenir d'autres procédures

Voici ce que pourrait donner un exemple de programme :

```
fonction factorielle_1 (n: entier): entier
     variables
     r:entier = 1;
     i:entier;
     début
     pour i de 1 jusqu'à n
        r := r*i;
     fin pour
     retourner r
  fin fonction

fonction factorielle_2 (n: entier): entier
     début
     si n < 2
        retourner 1;
     sinon
        retourner n * factorielle_2(n-1);
  fin fonction

programme début
    variables
    t: tableau[1..12] entier;
    i: 1..12;
    i := 1;
    tant que i < 12
       t[i] :=factorielle_1(i);
       i++;
    fin tant que
fin programme
```

Les seules commandes de code à trois adresses possibles sont les suivantes :

- $x := y$ `op` z
  Instruction d'affectation où *op* est un opérateur binaire arithmétique ou logique
- $x :=$ `op` y
  Instruction d'affectation où *op* est un opérateur unaire arithmétique, logique, de décalage et de conversion
- $x := y$
  Instruction de copie où la valeur de y est affectée à x
- *Label* L
  Etiquette l'instruction qui suit par le label L

- `JUMP` L
  Branchement inconditionnel qui a pour effet de faire exécuter ensuite l'instruction étiquetée par L
- `IF` x *op* y `JUMP` L
  Branchement conditionnel qui a pour effet de faire exécuter ensuite l'instruction étiquetée par L si la relation x *op* y est satisfaite. Où *op* est un opérateur relationnel ($<$, $>$, $\leq$, ...)
- Instructions d'appel de procédures.
  - `PARAM` x
    Passe un paramètre par valeur à une procédure
  - `CALL` p, n
    Apel de la procédure p qui prendra en compte n paramètres
  - `RETURN` y
    La procédure renvoie la valeur y
  - `FUNC` p
    Début de la procédure p
- Les instructions d'affectation de variables indicées
  - $x := y[i]$
    Où $y[i]$ désigne l'emplacement situé à i unités au-delà de la cellule y
  - $x[i] := y$
- Les instructions d'affectation d'adresses et pointeurs
  - $x := \&y$
    Où $\&y$ désigne l'adresse de y
  - $x := *y$
    Où $*y$ désigne la valeur pointée par y
  - $*x := y$

1 Questions

1. Ecrire une spécification pour le langage source que vous avez choisi. Une grammaire en forme de Backus-Naur est demandée.
   (a) Donner les conventions retenues pour l'écriture des commentaires, des déclarations, des expressions et des instructions. Donner des exemples dans la documentation.
   (b) Ecrire la grammaire en exploitant la précédences des opérateurs pour forcer une analyse LR implémentable avec `CUP`.
2. Ecrire en `CUP/JFLEX` un analyseur syntaxique pour ce langage :
   (a) Produire les messages d'erreur liés à une mauvaise utilisation des conventions syntaxiques.
   (b) Faire en sorte que le compilateur puisse poursuivre l'analyse après la détection d'une erreur de syntaxe.
   (c) Construire pour chaque expression le type associé.
   (d) Pour chaque déclaration de variable, dont les tableaux$^{(*)}$, d'enregistrement$^{(**)}$et de procédure$^{(***)}$, enregistrer dans une table de symboles les informations nécessaires à l'exploitation de l'identificateur (type, taille, décalage).
   (e) Construire pour chaque bloc, une table des symboles locale. Les déclarations de chaque bloc auront une portée que l'on précisera dans la documentation.
3. Ajouter à cet analyseur, un mécanisme de production de code à trois adresses implémentés sous la forme de quadruplets.
   (a) Implémenter les quadruplets qui contiendront les codes à trois adresses.
   (b) Pour les expressions arithmétiques et logiques, produire le code à trois adresses.

(c) Pour les instructions, produire le code à trois adresses

(d) Pour les expressions sur les tableaux$^{(*)}$, les enregistrements$^{(**)}$et les appels de procédures$^{(***)}$, produire le code à trois adresses

(e) Remplacer les variables déclarées dans le programme source par leur emplacements en mémoire statique. On utilisera $M[i]$ pour désigner l'emplacement mémoire décalée de i unités au-delà du début de la mémoire statique.