TP - 9/10

22/23 octobre 2014

On commence à travailler avec la structure du projet. Une partie du compilateur est déjà écrite (le *lexing* et le *parsing*). Le code existant permet donc d'extraire *l'arbre de syntaxe abstraite* d'un programme C- -. L'objectif de ce TP est d'exploiter ce code pour afficher correctement cet arbre sous la forme d'un docuement XML. Vous pourrez ensuite visualiser l'arbre et naviguer dans l'arbre en utilisant un navigateur web (par exemple).

Exercice - 1 *Premier contact*

1- Récupérer l'archive à cette adresse et décompressez-la (`$ tar -xf tp9.tar.gz`). Beaucoup de fichiers existent déjà. Lisez la description de la structure du programme ici et là (la page parle de x86-32 ne faites pas attention). Notez que j'ai ajouté le fichier `cprint.ml` à l'archive : il contiendra les fonctions d'affichage de l'arbre de syntaxe abstraite. Le second fichier qui nous intéressera plus particulièrement est `cparse.ml`. Il contient le type des listes de déclarations (représentant l'arbre de syntaxe abstraite).
2- Commencez par écrire le corps de la fonction `print_listFun` dans le fichier `cprint.ml` qui affiche le nom des différentes fonctions avec leurs arguments. L'output doit ressembler à

```
add (x, y)
sub (x, y)
print_tab (tab, size)
```

Exercice - 2 *Module `format`*

1- En vous aidant de la documentation du module format ainsi que de ce guide écrivez une fonction `format_list` qui affiche des listes imbriqués de string en indentant correctement. Le type des listes imbriquées que vous utiliserez doit être `type listImbr = F of string | I of listImbr list`.

Exercice - 3 *AST vers XML*

XML est un langage générique utilisé massivement dans différents domaines. Par exemples pour écrire des pages internets (XHTML), des dessin vectoriel (SVG), des textes formattés (OpenDocument), etc.

Dans cet exercice, vous devez générer un document XML représentant l'arbre de syntaxe abstraite d'un programme. En XML, on représente un noeud d'un arbre en utilisant des tags : `<tag>contenu</tag>`. Le contenu de ce noeud peut être un autre noeud XML ou du texte. Vous devez faire attention à bien refermer les balises (les balises *fermantes* commencent par un "/"). Vous devez refermer les balises en suivant l'ordre (inverse) d'ouverture. Par exemple `<a><b>Coucou</a></b>` est mal formé. Pour utiliser les symboles <,> et & dans du texte vous devez utiliser respectivement &lt, &lg et &amp.

Un noeud XML peut avoir des *attributs* (en plus de som nom) :

`<chapitre class="bibliographie">contenu</chapitre>`.

1- Ecrivez le corps de la fonction `print_locator` du fichier `cprint.ml` qui affiche les attributs d'un `locator` (cf. premier type dans le fichier `error.ml`). L'output attendu est de la forme :

```
file="tests/cat.c" first-line= "26" first-column="8" last-line="26" last-column="9"
```

2- Ecrivez le corps de la fonction `print_arbreSyntaxe` qui affiche l'arbre en XML indenté correctement (merci `Format`). Un exemple de document est donné dans la figure 1.
3- Modifiez `main.ml` pour qu'il accepte l'option `-p` qui a pour effet d'afficher l'arbre de syntaxe abstraite en XML. Vous pouvez maintenant enregistrer votre document XML (`$ ./mcc -p Exemple/cat.c > doc.XML`) et l'ouvrir avec un navigateur internet.

Exercice - 4 *(Bonus) ASCII art - c'est (aussi) joli*

1- Ecrivez une autre version de `print_arbreSyntaxe` qui affiche l'arbre en ASCII art. Ajoutez également une autre option dans `main.ml`. Un exemple de bel arbre est donné dans la figure 2, un autre figure 3. Laissez libre cours à votre créativité et imagination ;)

Fichier source :

```
int
main (int argc, char **argv)
{
  int a;
  return a + 1;
}
```

Rendu :

```
<cfun name="main" file="tests/test.c" first-line="2" first-column="0"
      last-line="2" last-column="4">
  <args>
    <cdecl name="argc" file="tests/test.c" first-line="2" first-column="10"
           last-line="2" last-column="14"/>
    <cdecl name="argv" file="tests/test.c" first-line="2" first-column="23"
           last-line="2" last-column="27"/>  </args>
  <cblock
    file="tests/test.c" first-line="3" first-column="0" last-line="6"
    last-column="1">
    <cdecl name="a" file="tests/test.c" first-line="4" first-column="6"
           last-line="4" last-column="7"/>
    <creturn
      file="tests/test.c" first-line="5" first-column="2" last-line="5"
      last-column="14">
      <add
        file="tests/test.c" first-line="5" first-column="9" last-line="5"
        last-column="14">
        <var name="a" file="tests/test.c" first-line="5" first-column="9"
             last-line="5" last-column="10"/>
        <cst value="1" file="tests/test.c" first-line="5" first-column="13"
             last-line="5" last-column="14"/></add></creturn></cblock>
  </cfun>
```

FIGURE 1 – Rendu attendu pour un programme simple

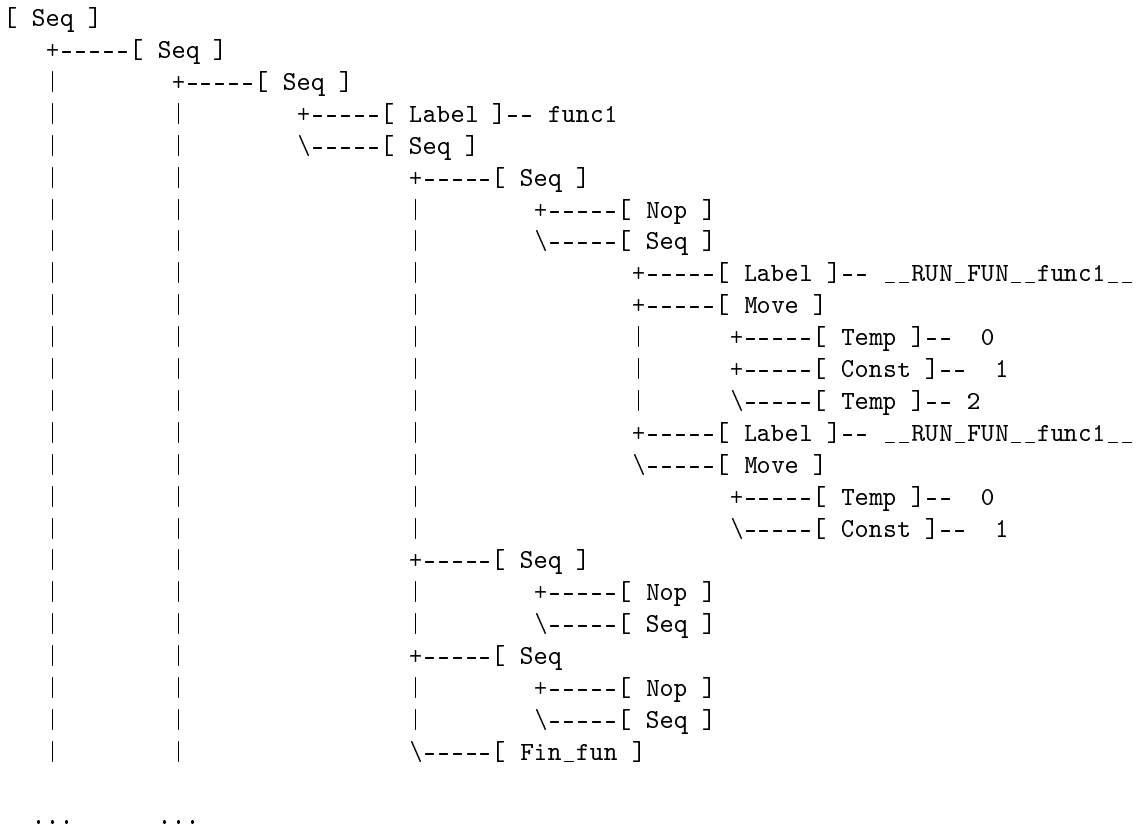

FIGURE 2 – Exemple de rendu attendu pour l'arbre en ASCII

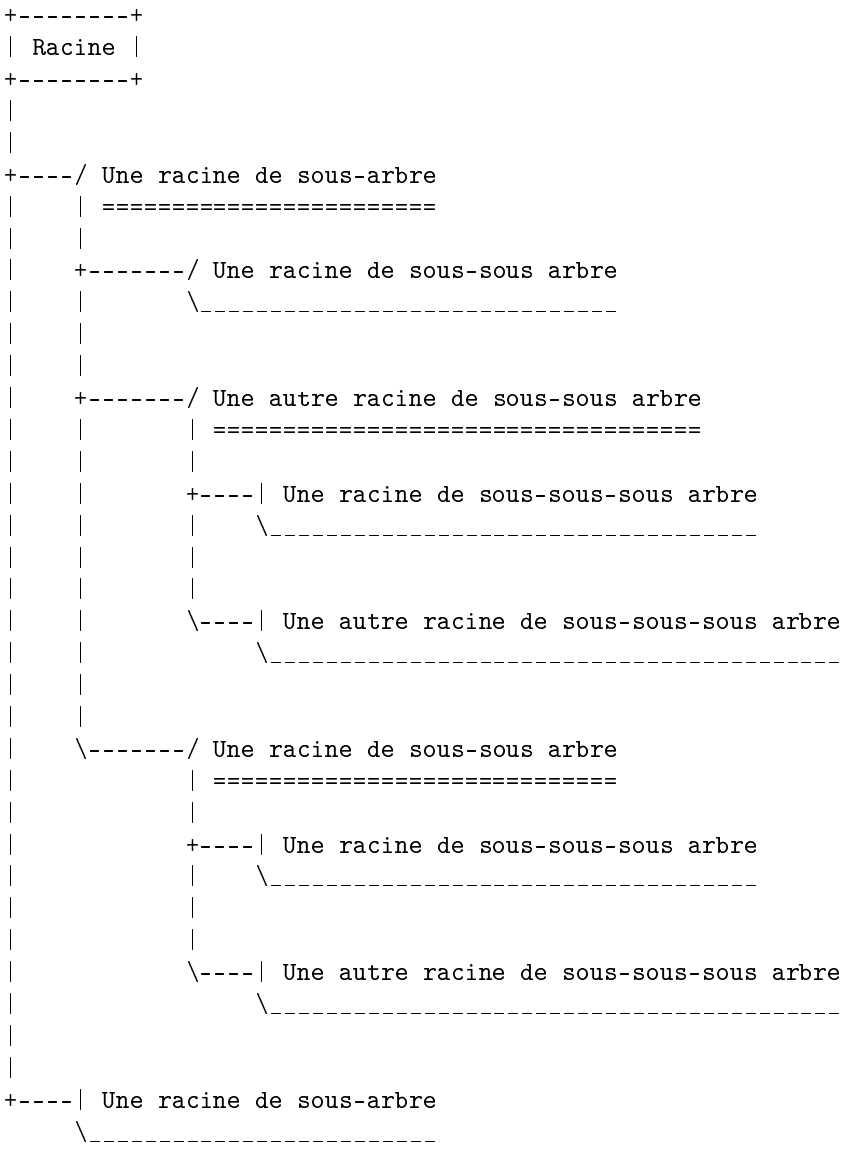

FIGURE 3 – Autre exemple de rendu pour l'arbre en ASCII