TP03 - TABLE DES SYMBOLES ET ANALYSE SÉMANTIQUE

1. Objectif

L'objectif de ce TP est de constuire pour un programme donné, la table des symboles lui correspondant et de réaliser une série de vérifications sémantiques, qui utilisent cette table. La table est remplie et consultée lors d'un parcours descendant de l'arbre abstrait qui a été constuit lors du TP précédent. A l'issue du parcours de l'arbre abstrait, plusieurs tables des symboles ont en fait été créées. Une table globale, qui contient les variables globales et les fonctions et autant de tables locales que le programme comporte de fonctions.
Une table rassemble toutes les informations utiles concernant les variables et les fonctions du programme.

- Pour toute variable, elle garde : son nom, son type, sa portée et son adresse en mémoire.
- Pour toute fonction elle garde : son nom, sa portée, le nom et le type de ses arguments, ainsi que leur mode de passage, et, éventuellement le type du résultat qu'elle fournit.

2. Le package `ts`

Il comporte trois classes.
La classe `TsItemVar` qui correspond à une entrée de type variable dans la table des symboles.
Elle définit les variables d'instance suivantes :

- `String identif` : le nom de la variable.
- `int taille` : la taille mémoire occupée par la variable. Elle est égale à un pour des variables simples et, pour les tableaux, à la taille de ces derniers.
- `Ts portee` : la portée de la variable, représentée par la table à laquelle appartient la variable.
- `boolean isParam` : indique si la variable est une variable (locale ou globale) ou un paramètre.
- `int adresse` : adresse relative de la variable. La première variable introduite dans la table a pour adresse 0, la seconde 0 + la taille de la première variable...

La classe `TsItemFct` qui correspond à une entrée de type fonction dans la table des symboles.
Elle définit les variables d'instance suivantes :

- `String identif` : le nom de la fonction.
- `int nbArgs` : son nombre d'arguments.
- `Ts table` table des symboles locale à la fonction.
- `SaDecFonc saDecFonc` :le nœud de l'arbre abstrait correspondant à la déclaration de la fonction.

La classe `ts` qui correspond à une table de symboles.
Elle définit les variables d'instance suivantes :

- `Map< String, TsItemFct> fonctions` : un dictionnaire associant à un identificateur de fonction son entrée.
- `Map< String, TsItemVar> variables` : un dictionnaire associant à un identificateur de variable son entrée.
- `int adrVarCourante` : l'espace mémoire occupé par les variables. Il augmente au fur et à mesure que l'on ajoute des variables dans la table.
- `int adrArgCourant` : l'espace mémoire occupé par les variables. Il augmente au fur et à mesure que l'on ajoute des paramètres dans la table.

Les principales méthodes de l'interface de la classe `Ts` sont les suivantes :

- `TsItemVar addVar(String identif, int taille)` ajoute une variable à la table des symboles et retourne l'entrée créée (une instance de la classe `TsItemVar`)
- `TsItemVar addParam(String identif)` ajoute un paramètre à la table des symboles et retourne l'entrée créée (une instance de la classe `TsItemVar`). On pourra remarquer que dans les deux cas précédent, une instance de la même classe est retournée. C'est la variable d'instance `isParam` qui permet de distinguer une variable d'un paramètre.

- `TsItemFct addFct(String identif, int nbArgs, Ts table, SaDecFonc saDecFonc)` ajoute une fonction à la table des symboles et retourne l'entrée créée (une instance de la classe `TsItemFct`).
- `TsItemVar getVar(String identif)` Retourne l'entrée correspondant à la variable ou au paramètre `identif` si elle existe et `null` sinon.
- `TsItemFct getFct(String identif)` Retourne l'entrée correspondant à la fonction `identif` si elle existe et `null` sinon.

3. Vérifications sémantiques en L

3.1. **Variables.**

Lors de la déclaration, vérifier que :
- Il n'y a pas deux variables identiques déclarées dans une même portée
  - Les variables locales et les arguments peuvent avoir le même nom qu'une variable globale.
  - Il n'est pas possible de déclarer une variable locale qui a le même nom qu'un argument.
- Un tableau est toujours une variable globale

Lors de l'appel dans une affectation ou expression, vérifier que :
- Toute variable utilisée est déclarée en tant que (recherche dans l'ordre) :
  - (1) Variable locale ou (2) argument de fonction ou (3) variable globale.
- Les tableaux ne peuvent jamais être utilisés sans indice
- Les entiers ne peuvent jamais être indicés
- Aucune conversion ou coercition n'est possible
- Il n'y a pas de vérification de dépassement des bornes du tableau

3.2. **Fonctions.**

Lors de la déclaration, vérifier que :
- Il n'y a pas deux fonctions identiques déclarées à des endroits différents
  - Il n'y a pas de *polymorphisme* en L, c'est-à-dire, deux fonctions sont identiques si leurs identificateurs sont identiques (indépendamment du nombre de paramètres)

Lors de l'appel dans une instruction ou expression, vérifier que :
- Toute fonction appelée doit être déclarée *avant* dans le programme
- Le nombre d'arguments réels passés à la fonction appelée est identique au nombre d'arguments formels dans la déclaration
- Il existe une fonction sans arguments qui s'appelle `main`

Votre compilateur **ne doit pas** vérifier :
- Le type de retour d'une fonction : c'est toujours un entier
- Le type d'un argument : c'est toujours un entier
- Toute branche d'exécution contient une instruction `retour`

4. Parcours de l'arbre abstrait

Le parcours de l'arbre abstrait est réalisé par l'intermédiaire du visiteur `SaDepthFirstVisitor`. Ce dernier permet d'effectuer un parcours en profondeur de l'arbre abstrait. Il définit une méthode `visit` pour chaque type de nœud de l'arbre abstrait.
En partant de ce visiteur, vous redéfinierez les méthode suivantes, qui définissent ou utilisent des fonctions, variables ou paramètres.
- `Void visit(SaDecVar node)`
- `Void visit(SaDecTab node)`
- `Void visit(SaDecFonc node)`
- `Void visit(SaVarSimple node)`
- `Void visit(SaVarIndicee node)`
- `Void visit(SaAppel node)`

A tout moment du parcours de l'arbre, il faut avoir accès à la table symboles globale et à la table des symboles locale (si on est en train de parcourir des nœuds faisant partie de la définition d'une fonction).

Il est aussi nécessaire, à tout moment, de savoir si on est dans un contexte global (hors de toute fonction) dans un contexte local (dans le corps d'une fonction) ou dans un contexte paramètre (dans la liste des paramètres d'une fonction).