Table des symboles

Introduction :

La table des symboles est utilisé au cours de l'analyse sémantique, troisième phase de la compilation qui fait suite à l'analyse syntaxique effectué dans les fichiers Yacc. Au cours de cette phase, l'arbre syntaxique (ayant été créé pendant l'analyse syntaxique) est parcouru et on vérifie à chaque instruction qu'il n'y a pas d'erreurs de typage ou d'évaluation. C'est par exemple au cours de cette phase qu'on doit détecter des erreurs lorsqu'on essaye d'affecter une mauvaise valeur à une variable (ex : integer i = 'a';) , ou bien qu'on affecte une valeur à une variable non définit, ou encore qu'on utilise une fonction avec des mauvais arguments. Pour connaître les type des variables définit, des fonctions ou des méthodes on doit au préalable les avoirs stocké quelque part, c'est le rôle de la table des symboles.

Utilisation :

On utilisera deux tables des symboles pendant l'analyse sémantique. Une table qui stockera les fonction/procédures et les méthodes des classes, et une autre qui stockera uniquement les variables. On utilisera pour cela uniquement les 4 classes ci dessous :

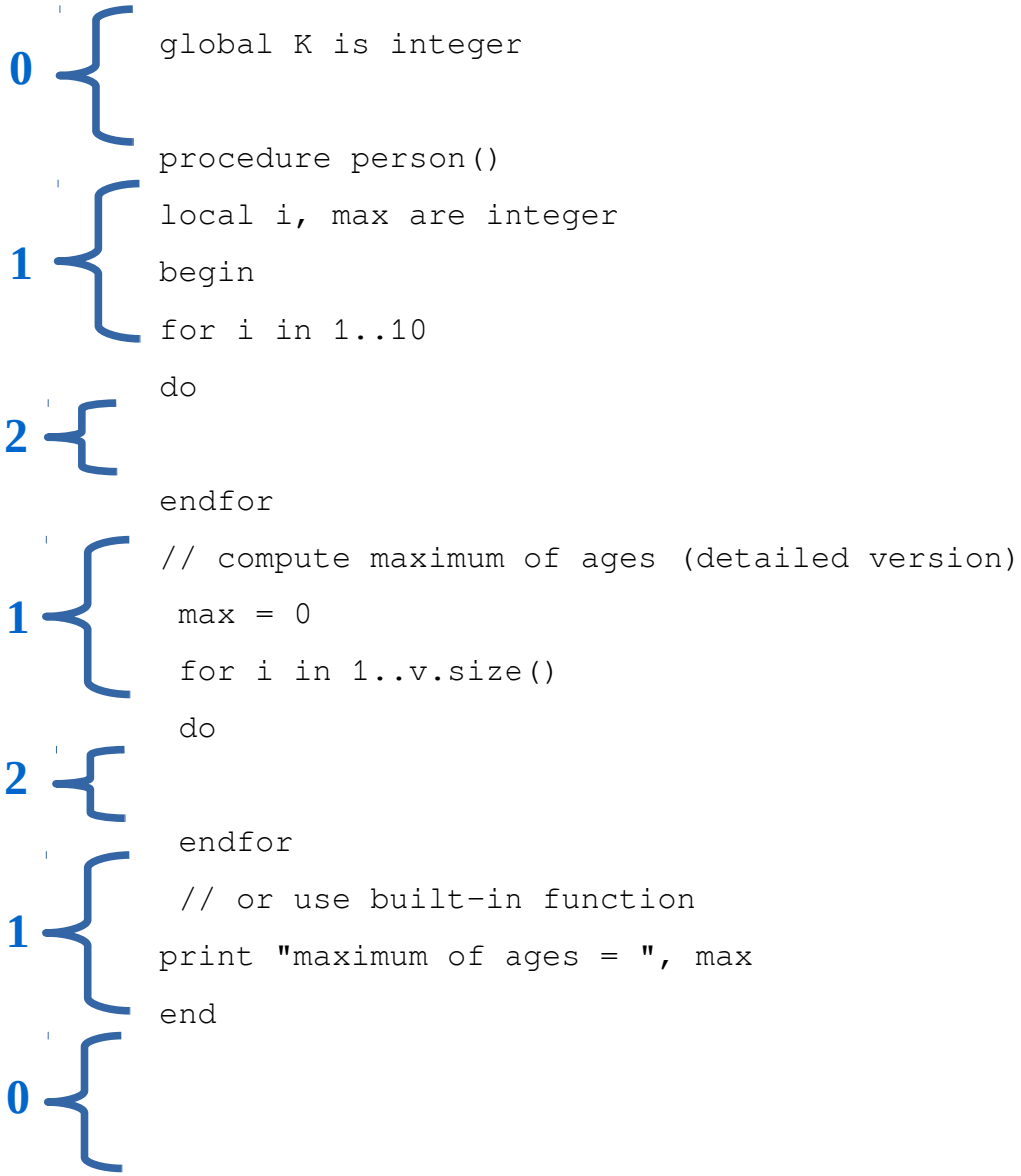
- `variable`
- `function`
- `hashTable<function>`
- `scopeHashTable`

Une `variable` possède 3 attributs : un identifiant, un type et une portée.

L'identifiant et le type sont des string et la portée est un entier non signé.

Une portée qualifie l'endroit dans le code ou a été déclarée la variable. La porté est initialement de 0. À chaque nouveaux bloc (boucle, déclaration de fonction etc...) la portée est incrémentée de 1. À chaque fin de partie (fin de boucle, fin de déclaration, de procédure etc …) la portée est décrémentée de 1.

Exemple :



Une `function` possède 4 attributs, un identifiant (string), une liste de paramètre (liste des types des paramètres : `vector<string>`), un type de retour et un dernier attribut `_class` qui devra définir la classe à laquelle la méthode appartient, ou une chaîne de caractère vide si il ne s'agit pas d'une méthode.

Une `hashTable<function>` est la table qui sera utilisée pour stocker les déclarations de fonction/procédures et méthodes. Son constructeur prend une taille en paramètre (50 semble une taille raisonnable). Pour ajouter une nouvelle déclaration à cette table, on créer un objet `function` et on utilise la méthode `addElement` qui prend en premier paramètre la `function` à ajouter et en second paramètre son identifiant . Pour savoir si une fonction/procédure méthode est

définit on utilise la méthode `contains` qui prend les même paramètres que `addElement` et qui renvoi vrai si elle a déjà été définit et faux sinon.

Une `scopeHahTable` est la table qui sera utilisée pour stocker les variables. Son constructeur prend également une taille en paramètre. Pour ajouter une variable il faut déclarer une instance de `variable` et utiliser la méthode `addElement` qui prend en paramètre cette instance de `variable`. Lorsqu'on entre dans une nouvelle boucle ou une nouvelle déclaration de fonction/procédure/méthode, on doit utiliser la méthode `incScope()`, qui incrémente la portée courante. De même, lorsqu'on sort d'une boucle ou d'une déclaration de fonction/procédure/méthode, on doit utiliser la méthode `decScope()`, qui supprime tous les éléments déclarés dans cette partie, et qui décrémente la portée courante. Pour connaître le type d'une variable, on peut utiliser la méthode `contains(string)` qui prend un identifiant(string) en paramètre et qui retourne vrai s'il existe une variable avec cet identifiant ayant déjà été déclarée. Pour connaître le type d'une variable on peut utiliser la méthode `get_type(string)` qui prend en paramètre un identifiant(string) et qui retourne le type de la dernière variable déclarée avec cet identifiant (si il existe plusieurs variables avec le même identifiant dans plusieurs portées différentes). Retourne une exception (string) si cet identifiant n'a pas été trouvé dans la table des symboles.