Raport du projet de compilation

CHEMINADE Dorian
SENEL Yasin

7 décembre 2013

0.1 Spécification du langage

Commentaires

– commentaire sur une ligne : __COMMENT__ <votre commentaire ici>

– commentaire sur plusieurs lignes : __COMMENT__(<debut de votre long commentaire>
<fin de votre long commentaire>)

Types Simples

– entier signé sur 1 octet (-128..127) : Integer <name> ;

– entier signé sur 2 octets (-32768..32767) : BigInteger <name> ;

– entier non signé sur 1 octet (0..255) : UnsignedInteger <name> ;

– entier signé sur 2 octets (0..65535) : UnsignedBigInteger <name> ;

– booléen : Boolean <name> ;

– caractère (sur 4 octets) : Character <name> ;

– nombre réel ($1.5 * 10^{-45}$..$3.4 * 10^{38}$) : Real <name> ;

– énumération non gérée.

Types Complexes

– intervalles : Non gérés

– string (de (1..65535) caractères)) : String <name> ;

– array : Non géré

– pointer :
déclaration de pointeur : <type> <name> =-> <var>
obtenir l'adresse du pointeur : name
obtenir le contenue du pointeur : name->

Structure de Contrôle

– switch
SWITCH
CASE (<condition>_1>)
<instructions>
...
CASE (<condition_n>)
<instructions>
END_SWITCH

Boucles

– while :
WHILE (<condition>) DO
<instructions>
OD

0.2 Choix d'implémentation

0.2.1 Parseur

On utilise un parseur CUP pour parser notre grammaire. A chaque règle dans la grammaire on crée un noeud que l'on fait remonter. Les érreurs de syntaxe sont traitées de façon simple au niveau des premières règles de la grammaire, et le parser continue l'analyse du code.

0.2.2 Arbre de syntaxe abstraite

Nous avons utilisé un arbre de syntaxe abstraite pour produire du code 3 adresses. Nous avons choisie cette implémentation car elle reflète bien la structure du code source, permet de gérer les environnements et de contrôler les types. On a différent type de noeud qui corresponde tous a un élément dans le programme (exemple : NodeWhile, NodeVariable, NodeArithmetic...). Chaque noeud implémente une interface (Node) qui nous fournit une méthode GetTac. GetTac permet de générer le code a 3 adresse. Nous avons choisit cette solution plutôt que le modele Visiteur pour ne pas avoir a créer trop de sous-classes.

0.2.3 Table des symboles

Pour éviter la déclaration multiple de variable nous avons décidé d'utiliser une table des symboles. A chaque déclaration de variable on ajoute la variable dans la table. S'il y a plusieurs déclaration de variable on envoie une erreur (NodeError) qui indique la ligne et la colonne de l'erreur. Ainsi il est impossible de déclarer une variable plusieurs fois ou d'affecter un contenue a une variable sans l'avoir déclaré avant. De plus cette table des symboles aurai pu nous servir a vérifier les types de variable mais par manque de temps nous n'avons pas pu effectuer cette partie. Notre table de symboles ne gere pas les différents contextes du code (blocs, boucles while, ...), les variables sont toutes vues comme des variables globales.

0.3 Problèmes et solutions

La détection des érreurs de syntaxe de façon précise provoquait des conflits dans le parser, on a donc opté pour une solution plus generale qui ne décrit pas l'érreur rencontrée mais qui indique sont emplacement. Par manque de temps notre table de symboles ne gere pas les différents contextes du code (blocs, boucles while, ...), les variables sont toutes vues comme des variables globales.