Compilateur pour langage `while`
Devoir de programmation, partie I : Analyse
Université d'Orléans, Master 1 Informatique, 2012-2013

L'objectif de ce devoir est de réaliser la phase d'analyse pour la compilation des programmes d'un langage simplifié. La production de code sera réalisée dans un second devoir. Vous devrez mettre en place les phases d'analyse lexicale, syntaxique et sémantique pour ce langage. La partie 1 décrit la syntaxe du langage, la partie 2 donne les règles de typage nécessaires à la phase d'analyse sémantique et la partie 3 décrit le travail à réaliser.

1 Syntaxe

Expressions. Les expressions du langage sont divisées en deux catégories, expressions arithmétiques A et expressions booléennes B.

$$A \ ::= \ n \mid d \mid x \mid x[A] \mid \mathit{unop}\ A \mid A\ \mathit{binop}\ A \mid (A)$$

$$B \ ::= \ \texttt{true} \mid \texttt{false} \mid A\ \mathit{cmp}\ A \mid !\ B \mid B\ \&\&\ B \mid B \mid\mid B \mid (B)$$

où n dénote une constante entière, d une constante décimale, x un identificateur et

$$\mathit{unop} \in \{-\} \qquad \mathit{binop} \in \{+, -, *, /\} \qquad \mathit{cmp} \in \{<, >, <=, >=, =, <>\}$$

Un identificateur est une suite de caractères alpha-numériques commençant par une lettre minuscule. Une expression $x[A]$ représente l'accès à l'indice dénoté par A du tableau x. Les règles de priorités sont les règles usuelles des opérateurs arithmétiques et booléens.

Types. Les types du langage sont donnés par la règle $T ::= \texttt{int} \mid \texttt{decimal} \mid \texttt{array}(T)$. Il n'est pas nécessaire de distinguer un type pour les expressions booléenes.

Instructions. Les instructions du langage sont définies de la manière suivante

$$\begin{aligned} LHS \ &::= \ x \mid x[A] \\ \\ I \ &::= \ \mid \texttt{return} \mid \texttt{return}\ A \\ &\quad\ \mid LHS := A \\ &\quad\ \mid LHS = \texttt{new}\ T[A] \mid \texttt{free}(A) \\ &\quad\ \mid f(A, \ldots, A) \mid LHS := f(A, \ldots, A) \end{aligned}$$

Une instruction peut être une affectation de la forme $LHS := A$ où LHS représente l'emplacement mémoire ou sera stockée la valeur de l'expression A. La partie gauche LHS peut être une variable ou une case d'un tableau. L'instruction $LHS := \texttt{new}\ T[A]$ représente l'allocation d'un tableau de type T dont la taille est donnée par le résultat de l'évaluation de A. L'instruction $\texttt{free}(A)$, où A représente l'adresse d'un tel tableau, permet de récupérer la mémoire ainsi allouée. On dispose également d'instructions d'appels de procédure/fonction et de retour.

Statements. Les statements du langage sont définis à partir des instructions de la manière suivante

$$\begin{array}{rcll} S & ::= & \mid I & \\ & & \mid \texttt{if}\ b\ \texttt{then}\ S\ \texttt{else}\ S & \\ & & \mid \texttt{while}\ b\ \texttt{do}\ S & \\ & & \mid \texttt{begin}\ VDECL_1 \ldots VDECL_m\ S_1; \ldots; S_n\ \texttt{end} & m \geq 0,\ n \geq 1 \end{array}$$

Un statement peut être une instruction, une conditionnelle, une boucle ou un block. Un block est une suite de déclaration de variables locales, données par la règle ci-dessous, suivie d'une séquence de statements.

$$VDECL ::= T\ x_1, \ldots, x_n;$$

Programmes. Un programme est constitué d'une suite de déclarations de procédures et fonctions et d'un statement dénotant le corps du programme (équivalent de la fonction *main* en Java).

$$\begin{array}{rcll} P & ::= & PDECL_1 \ldots PDECL_n\ S & n \geq 0 \\ PDECL & ::= & \mid \texttt{function}\ f(x_1 : T_1, \ldots, x_n : T_n) : T\ \texttt{is}\ S & n \geq 0 \\ & & \mid \texttt{procedure}\ f(x_1 : T_1, \ldots, x_n : T_n)\ \texttt{is}\ S & n \geq 0 \end{array}$$

2 Typage

Typage des expressions On note Γ une suite d'éléments $x : \tau$. Intuitivement, Γ, que l'on appelle environnement de typage, joue le rôle de table des symboles. L'environnement se lit de droite à gauche, i.e s'il y a plusieurs occurences d'une même variable, l'occurence courante est celle qui se trouve la plus à droite. On note $\Gamma(x)$ le type associé à cette occurence de x dans Γ. On définit ci-dessous les règles d'inférence des relation $\Gamma \vdash_a e : \tau$ et $\Gamma \vdash_b B$ qui caractérisent respectivement le bon typage d'une expression arithmétique et

d'une expression booléenne.

$$\frac{}{\Gamma \vdash_a n : \texttt{integer}} \qquad \frac{}{\Gamma \vdash_a d : \texttt{decimal}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash_a x : \tau}$$

$$\frac{\Gamma \vdash_a A : \texttt{integer} \qquad \Gamma(x) = \texttt{array}(\tau)}{\Gamma \vdash_a x[A] : \tau}$$

$$\frac{\Gamma \vdash A : \tau \qquad \tau \in \{\texttt{integer}, \texttt{decimal}\}}{\Gamma \vdash_a \mathit{unop}\ A : \tau}$$

$$\frac{\Gamma \vdash_a A_1 : \tau \qquad \Gamma \vdash_a A_2 : \tau \qquad \tau \in \{\texttt{integer}, \texttt{decimal}\}}{\Gamma \vdash_a A_1\ \mathit{binop}\ A_2 : \tau}$$

$$\frac{}{\Gamma \vdash_b \texttt{true}} \qquad \frac{}{\Gamma \vdash_b \texttt{false}} \qquad \frac{\Gamma \vdash_a A_1 : \tau \qquad \Gamma \vdash_a A_2 : \tau \qquad \tau \in \{\texttt{integer}, \texttt{decimal}\}}{\Gamma \vdash_b A_1\ \mathit{cmp}\ A_2}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash_b ! B} \qquad \frac{\Gamma \vdash B_1 \qquad \Gamma \vdash B_2}{\Gamma \vdash_b B_1\ \&\&\ B_2} \qquad \frac{\Gamma \vdash B_1 \qquad \Gamma \vdash B_2}{\Gamma \vdash_b B_1\ \&\&\ B_2}$$

Typage des instructions On définit maintenant les règles permettant de vérifier qu'une instruction est correctement typée. Pour cela, on ajoute à l'environnement Γ, un environnement Δ qui associe un schéma à chaque symbole de fonction. Les règles sont maintenant de la forme

$$\Gamma; \Delta; t \vdash_i I$$

qui indique que l'instruction I est correctement typé pour les environnement Γ et Δ pour un type de retour t où t est soit $\bullet$ dans le cas d'une procédure soit un type τ dans le cas

d'une fonction.

$$\frac{}{\Delta, \Gamma; \bullet \vdash_i \texttt{return}} \qquad \frac{\Gamma \vdash_a e : \tau}{\Delta; \Gamma; \tau \vdash_i \texttt{return}\ e}$$

$$\frac{\Gamma \vdash_a LHS : \tau \qquad \Gamma \vdash_a A : \tau}{\Delta; \Gamma; t \vdash_i LHS := A} \qquad \frac{\Gamma \vdash_a LHS : \texttt{array}(T) \qquad \Gamma \vdash_a A : \texttt{integer}}{\Delta; \Gamma; t \vdash_i LHS := \texttt{new}\ T\ [A]}$$

$$\frac{\Gamma \vdash_a A : \texttt{array}(\tau)}{\Delta; \Gamma; t \vdash_i \texttt{free}(A)}$$

$$\frac{\Delta(f) = \tau_1 \times \ldots \times \tau_n \qquad \Delta; \Gamma; t \vdash_a A_i : \tau_i,\ i = 1, \ldots, n}{\Delta; \Gamma; t \vdash_i f(A_1, \ldots, A_n)}$$

$$\frac{\Delta(f) = \tau_1 \times \ldots \times \tau_n \to \tau \qquad \Gamma \vdash_a : LHS : \tau \qquad \Delta; \Gamma; t \vdash_a A_i : \tau_i,\ i = 1, \ldots, n}{\Delta; \Gamma; t \vdash_i LHS := f(A_1, \ldots, A_n)}$$

Typage des statements Le typage des statements suit le même schéma que celui des instructions, on définit la relation $\vdash_s$ par les règles suivantes.

$$\frac{\Delta; \Gamma; t \vdash_i I}{\Delta; \Gamma; t \vdash_s I}$$

$$\frac{\Gamma \vdash_b b \qquad \Delta; \Gamma; t \vdash_s S_1 \qquad \Delta; \Gamma; t \vdash_s S_2}{\Delta; \Gamma; t \vdash_s \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2} \qquad \frac{\Gamma \vdash_b b \qquad \Delta; \Gamma; t \vdash_s S}{\Delta; \Gamma; t \vdash_s \texttt{while}\ b\ \texttt{do}\ S}$$

$$\frac{\Delta; \Gamma, x_1^1 : T^1, \ldots, x_{k_1}^1 : T^1, \ldots, x_1^m : T^1, \ldots, x_{k_m}^m : T^m; t \vdash_s S_i \qquad i = 1, \ldots, n}{\Delta; \Gamma; t \vdash_s \texttt{begin}\ T^1\ x_1^1, \ldots, x_{k_1}^1; \ldots; T^m\ x_1^m, \ldots, x_{k_m}^m; S_1; \ldots; S_n \texttt{end}}$$

Typage des programmes

$$\Delta;\ ; \bullet \vdash_s S$$

$$\Delta; x_1 : \tau_1 \ldots; \tau_n; \bullet \vdash_s S \qquad \text{pour chaque } PDECL_i = \texttt{procedure}\ f(x_1 : \tau_1, \ldots, x_n : \tau_n)\ \texttt{is}\ S$$

$$\Delta; x_1 : \tau_1 \ldots; \tau_n; \tau \vdash_s S \qquad \text{pour chaque } PDECL_i = \texttt{function}\ f(x_1 : \tau_1, \ldots, x_n : \tau_n)\ : \tau\ \texttt{is}\ S$$

$$\frac{\begin{array}{rl}\Delta\ = & (\bigcup_1^n \{f : \tau_1 \times \ldots \times \tau_n \mid PDECL_i = \texttt{procedure}\ f(x_1 : \tau_1, \ldots, x_n : \tau_n)\ \texttt{is}\ S\}) \\ \cup & (\bigcup_1^n \{f : \tau_1 \times \ldots \times \tau_n \to \tau \mid PDECL_i = \texttt{function}\ f(x_1 : \tau_1, \ldots, x_n : \tau_n)\ : \tau\ \texttt{is}\ S\})\end{array}}{\vdash PDECL_1 \ldots PDECL_n\ S}$$

3 Travail à faire

Objectifs : Définir les différentes phases d'analyse pour le langage considéré ici, en utilisant `flex` et `bison`.

- Attention la description du langage n'est peut être pas adaptée à la reconnaissance syntaxique et il sera peut être nécessaire de redéfinir la grammaire pour lever les ambiguïtés.
- le système de type assure que les instructions `return` e et `return`, lorsqu'elles existent, sont légales. En revanche rien ne garantit qu'elles sont bien présentes dans tous les cas. Vous devrez proposer une solution pour régler ce problème, par exemple en modifiant les règles de typage.

Comportement de votre programme : votre programme devra

- prendre en entrée un fichier source et générer l'arbre de syntaxe correspondant (en utilisant une grammaire attribuée pour faire remonter l'information de typage). Il sera nécessaire de construire une table des symboles pour gérer le problème de portée des variables.
- générer en sortie une représentation graphique du fichier source au format `dot` de `graphviz`
- générer une erreur compréhensible si le programme est incorrect.

Constitution des groupes : groupes de 3 ou 4 étudiants.

Format de rendu : Une archive au format Nom1_Nom2_Nom3.`tar.gz` contenant les sources de votre programme, une série de fichiers sources de test un fichier README décrivant l'utilisation du programme, et un document au format `pdf` expliquant votre solution pour gérer le problème des instructions de retour absentes.

Remarque : Il vous est demandé d'apporter une attention particulière aux structures de données utilisées pour la représentation de l'arbre qui sera utilisé dans la phase de production de code.