Première partie, cahier des charges

29 août 2001

1 Compilation de votre projet

Votre projet doit se présenter sous forme d'une archive tar compressée (option "z" de tar), appelée *vos_noms*`.tgz` qui doit contenir un répertoire appelé *vos_noms* (exemple: `courant-filliatre.tgz`). Dans ce répertoire doivent se trouver les *sources* de votre programme (ne donnez pas les fichiers compilés). Quand on se place dans ce répertoire, la commande `make` doit créer votre compilateur, qui sera appelé `scalpac`. La commande `make clean` doit effacer tous les fichiers que `make` a engendré et ne laisser dans le répertoire que les fichiers sources.

2 Conventions lexicales de Scalpa

2.1 Blancs

Les caractères espace, retour chariot, tabulation sont considérés comme blancs. Ils sont pris en compte uniquement lors de l'analyse lexicale en tant que séparateurs et sont ignorés par la suite. Plus précisément, un *lexème* ne peut chevaucher un blanc : la présence d'un blanc permet donc de forcer la séparation entre deux lexèmes. Notez que deux lexèmes ne sont pas nécessairement séparés par un blanc. Par exemple 33*21 sera analysé comme une suite de trois lexèmes.

2.2 Commentaires

Les commentaires sont introduits avec les caractères `(*` et se terminent avec les caractères `*)`. Les commentaires sont traités de la même façon que les blancs par l'analyseur lexical. Les commentaires peuvent être imbriqués.

2.3 Identificateurs (*ident*)

Les identificateurs sont constitués d'une suite non vide de caractères dont le premier caractère est alphabétique et dont tous les autres sont alphanumériques, _, ou '. Leur longueur est arbitraire. Les mots clés de Scalpa ne pourront être des identificateurs.

2.4 Constantes entières (*int*)

Ce sont des suites non vides de chiffres.

2.5 Mots clés

Les mots clés sont les suivants : `int bool unit if then else true false not void while do program function var array return ref of begin end`

3 Syntaxe de Scalpa

La syntaxe de Scalpa que votre compilateur doit accepter est la suivante :

$$program \quad ::= \quad \texttt{program}\ ident\ vardecllist\ fundecllist\ instr$$

$$\begin{array}{rcl} vardecllist & ::= & \epsilon \\ & | & varsdecl \\ & | & varsdecl\ \texttt{;}\ vardecllist \end{array}$$

$$varsdecl \quad ::= \quad \texttt{var}\ identlist\ \texttt{:}\ typename$$

$$\begin{array}{rcl} identlist & ::= & ident \\ & | & ident\ \texttt{,}\ identlist \end{array}$$

$$\begin{array}{rcl} typename & ::= & atomictype \\ & | & arraytype \end{array}$$

$$\begin{array}{rcl} identlist & ::= & ident \\ & | & ident\ \texttt{,}\ identlist \end{array}$$

$$\begin{array}{rcl} atomictype & ::= & \texttt{unit} \\ & | & \texttt{bool} \\ & | & \texttt{int} \end{array}$$

$$arraytype \quad ::= \quad \texttt{array}\ \texttt{[}\ rangelist\ \texttt{]}\ \texttt{of}\ atomictype$$

$$\begin{array}{rcl} rangelist & ::= & int\ \texttt{..}\ int \\ & | & int\ \texttt{..}\ int\ \texttt{,}\ rangelist \end{array}$$

$$\begin{array}{rcl} fundecllist & ::= & \epsilon \\ & | & fundecl\ \texttt{;}\ fundecllist \end{array}$$

$$fundecl \quad ::= \quad \texttt{function}\ ident\ \texttt{(}\ arglist\ \texttt{)}\ \texttt{:}\ atomictype\ vardecllist\ instr$$

$$\begin{array}{rcl} arglist & ::= & \epsilon \\ & | & arg \\ & | & arg\ \texttt{,}\ arglist \end{array}$$

$$\begin{array}{rcl} arg & ::= & ident\ \texttt{:}\ typename \\ & | & \texttt{ref}\ ident\ \texttt{:}\ typename \end{array}$$

$$\begin{array}{rcl} instr & ::= & \texttt{if}\ expr\ \texttt{then}\ instr \\ & | & \texttt{if}\ expr\ \texttt{then}\ instr\ \texttt{else}\ instr \\ & | & \texttt{while}\ expr\ \texttt{do}\ instr \\ & | & lvalue\ \texttt{:=}\ expr \\ & | & \texttt{return}\ expr \\ & | & \texttt{return} \\ & | & ident\ \texttt{(}\ exprlist\ \texttt{)} \\ & | & \texttt{begin}\ sequence\ \texttt{end} \\ & | & \texttt{begin}\ \texttt{end} \end{array}$$

$$\begin{array}{rcl} sequence & ::= & instr\ \texttt{;}\ sequence \\ & | & instr\ \texttt{;} \\ & | & instr \end{array}$$

```
lvalue   ::= ident
           | ident [ exprlist ]

exprlist ::= expr
           | expr , exprlist

expr     ::= true
           | false
           | void
           | int
           | ident ( exprlist )
           | ident [ exprlist ]
           | ident
           | ( expr )
           | expr & expr
           | expr or expr
           | not expr
           | expr < expr
           | expr > expr
           | expr <= expr
           | expr >= expr
           | expr = expr
           | expr <> expr
           | expr + expr
           | expr * expr
           | expr / expr
           | expr - expr
           | expr ^ expr
           | - expr
```

4 Invocation du compilateur

`scalpac` [*options*] *fichier* compile le fichier *fichier*. La valeur de retour de la commande est 0 si tout s'est bien passé et sinon elle doit être supérieure ou égale à 1. La liste des options doit contenir au moins :

-lex-only : n'effectue que la phase d'analyse lexicale.

-dump-tokens : affiche la liste des tokens reconnus séparés par un retour à la ligne.

-parse-only : n'effectue que la phase d'analyse lexicale et la phase d'analyse syntaxique.

-dump-ast : affiche l'arbre de syntaxe abstraite.

-pretty-ast : affiche un texte de programme dont l'arbre de syntaxe abstraite est le même que celui du programme donné en argument. En particulier, si vous faîtes

```
scalpac -parse-only -pretty-ast toto.bla > titi.bla
scalpac -parse-only -pretty-ast titi.bla > tutu.bla
```

alors `titi.bla` et `tutu.bla` doivent être identiques. Cette option pourra ajouter des espacements et indenter le code intelligemment.

-help : affiche un message décrivant la façon d'invoquer le compilateur.

-version : affiche le numéro de version du compilateur et les noms des auteurs.

-debug n : affiche diverses information de déverminage du compilateur pendant son exécution. Si n vaut 0 (valeur par défaut) rien n'est affiché. Plus n est grand, plus le nombre d'informations est important (à vous de choisir le nombre de niveaux nécessaires).

Pour l'instant si ni `-lex-only` ni `-parse-only` ne sont spécifiés, `scalpac` doit échouer en indiquant que le reste de la chaîne de compilation n'est pas implanté. En cas d'erreur lexicale, `scalpac` doit afficher un message sur la sortie standard d'erreur indiquant la position dans le fichier original de l'erreur en respectant la syntaxe utilisée dans l'exemple suivant :

```
File "toto.bla", line 10, characters 24-29 : Lexical error.
```

De même, en cas d'erreur syntaxique on écrira :

```
File "titi.bla", line 277, characters 10-96 : Syntax error.
```