Feuille de TP n° 4 (séance 5) : langage Beaver

Nous allons travailler en TP sur un langage assez simple appelé BEAVER. Le but de ce TP est de réaliser son analyseur syntaxique.

1 Syntaxe concrète

La syntaxe concrète est donnée par une grammaire BNF (*Backus-Naur Form*).

Dans cette grammaire,

- les accolades indiquent la présence éventuelle de ce qui est entre accolades,
- les mots clefs du langages sont donnés entre guillemets (').
- IDENT dénote une chaîne de caractères commençant par une lettre minuscule et composée de lettres minuscules ou majuscules ainsi que du caractère '_' et du caractère '
- OP dénote l'un des opérateurs du langage : '+', '-', '*', '/', '&' (et logique), '|' (ou logique), '>', '>=', '<', '<=', '=', '<>' (négation de l'égalité).
- NUM dénote une constante entière (une suite fini de chiffres).
- STRING dénote une chaîne de caractères c'est à dire une suite de symboles différents du caractère '"' comprise entre deux '"'.
- CHAR dénote une chaine de caractère de la forme `'c'` où `c` est n'importe quel caratère différent de `'` ou la chaîne `\n`.
- BOOL dénote une constante booléenne c'est à dire 'true' ou 'false'

```
vardecl     ::=
                'var' IDENT {' :' IDENT} ' :=' bexp

fundecl     ::=
                IDENT '(' fields ')' {' :' IDENT} '=' bexp
                {'and' fundecl}

typedecl    ::=
                IDENT '=' typedef {'and' typedecl}

typedef     ::=
                IDENT
              | '{'fields'}'
              | 'array' 'of' IDENT

fields      ::=
                IDENT ' :' IDENT {',' fields}

assignable  ::= /* expressions assignables */
                IDENT                          /* variables*/
              | assignables '.' IDENT          /* accès à un champ */
              | assignables '['bexp']'         /* accès à un élément de tableau */

bexp        ::=
                assignable                     /* objet assignable */
              | 'nil'                          /* constante nil */
              | NUM                            /* constante entiere */
              | CHAR                           /* constante de caractère */
              | STRING                         /* constante de chaine */
              | BOOL                           /* constante booléenne */
              | IDENT '{'expfields'}'          /* construction d'enregistrement */
              | IDENT '[' bexp ']' 'of' bexp   /* construction de tableau */
```

	|	IDENT'(' {arguments} ')'	/* appel de fonction */
	|	bexp ';' bexp	/* sequencement */
	|	'if' bexp 'then' bexp { 'else' bexp }	/* conditionnelle */
	|	'while' bexp 'do' bexp 'done'	/* boucle while */
	|	'for' IDENT ' :=' bexp 'to' bexp 'do' bexp 'done'	/* boucle for */
	|	'let' fundecl 'in' bexp	/* déclaration de fonctions */
	|	vardecl 'in' bexp	/* déclaration de variable */
	|	'type' typedecl 'in' bexp	/* déclaration de types */
	|	bexp OP bexp	/* application d'opérateur */
	|	assignable ' :=' bexp	/* affectation */
	|	'(' bexp ')'	
	|	'begin' bexp 'end'	
arguments	::=		
		bexp {',' arguments }	
expfields	::=		
		IDENT '=' bexp { ',' expfields }	

Les caractères \t (tabulation), \n (retour chariot) et espace forment les séparateurs du langage Beaver.

Les commentaires du langages Beaver sont placés entre `/*` et `*/`. Dans un premier temps, on considérera que l'on ne peut pas imbriquer de commentaires.

2 Analyse syntaxique

Le répertoire `/ens/prieur/Licence/Prog2/LexYacc/Beaver` contient
- un lexeur pour le langage Beaver (fichier `lexer.mll`) ;
- un début de parseur (fichier `parser.mly`) ;
- la syntaxe abstraite (fichier `beaver.ml`) ;
- un fichier `test_syntaxe.ml` ;
- un `Makefile` pour compiler.

Travail à réaliser :
1. Observer `lexer.mll` et comprendre son fonctionnement.
2. Compiler et exécuter le programme. Quel est le sous-ensemble du langage reconnu ?
3. Étendre le parseur avec les opérateurs (OP). Compiler et tester.
4. Étendre le parseur avec le if. Compiler et tester.
5. Ajouter les appels et définition de fonctions. Compiler et tester.
6. Comment peut-on gérer les commentaires imbriqués ?
7. Comment faire pour indiquer le numéro de ligne dans les messages d'erreur ?
8. Compléter le parseur pour traiter tout le langage.