3 A. Licence

Implémentation d'un analyseur lexical

On se propose d'implémenter un analyseur lexical pour le mini-langage de programmation dont la syntaxe est décrite par la grammaire suivante :

$$I \rightarrow I; I \mid id = E \mid E? I \mid \{I\}$$

$$E \rightarrow id \mid num \mid E \text{ op } E$$

id désigne un identificateur constitué d'une séquence, éventuellement vide, de lettres majuscules et de chiffres décimaux, précédée par le caractère "\$"; ainsi, \$, \$23, \$MA, \$T12, \$5A7X ... sont des identificateurs.

num représente une constante entière non signée constituée d'une séquence non vide de chiffres hexadécimaux ([0 - 9A -]).

L'instruction E? I est considérée comme étant sémantiquement équivalente à l'instruction if (E) I du langage C.

Dans les expressions, *op* désigne l'un des opérateurs +, - ou %, avec % représentant le reste de la division entière (le modulo).

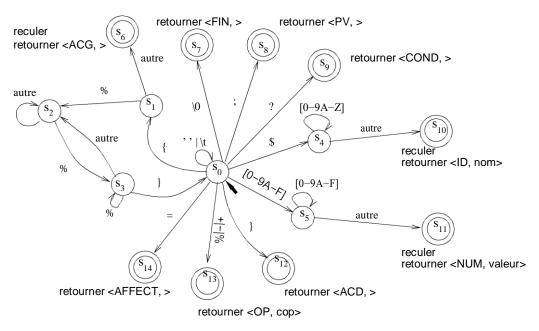
Dans le mini-langage, toute séquence de caractères comprise entre "{%" et "%}" est un commentaire.

Le tableau suivant donne une description des tokens du mini-langage.

Token	F IN	PV	AF F EC T	CON D	ACG	ACD	ID	N UM	OP	
Modèle	\0	;	=	? {		}	\$[0-9A-Z]*	$[0-9A-F]^{+}$	+ - %	
Attribut							nom	valeur	cop	

avec $cop \in \{PLUS, MOINS, MOD\}.$

Le DFA reconnaissant les différents tokens et incluant les actions aux niveaux des états finaux est :



La codification des caractères est donnée par la table suivante :

Caractère	\0	;	=	?	{	}	+ -	%	\$	[0 - 9]	[A – F]	[G-Z]	[\t]	autre
Code entier	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Travail à accomplir:

- 1. Implémenter l'analyseur lexical du mini-langage. Le programme doit, entre autres, inclure :
 - (a) la déclaration des tokens et de la variable globale d'attribut,
 - (b) la définition de la fonction car_suivan()
 - (c) la définition de la fonction $get_lexem()$ permettant de retourner le dernier lexème reconnu,
 - (d) la définition d'une fonction atoh() permettant de convertir une chaîne composée d'une suite de chiffres hexadécimaux en la valeur numérique correspondante; en d'autres termes, l'équivalent de la fonction atoi() mais pour des nombres hexadécimaux,
 - (e) la définition de la fonction token_suivan()
 - (f) la fonction principale mai() qui, par appels répétés à $token_suivant()$, obtient et affiche la liste des couples < token, attribut >.
- 2. Effectuer diverses exécutions sur des programmes sources avec ou sans commentaires, et avec ou sans erreurs lexicales.
- 3. Modifier l'analyseur pour permettre :
 - (a) la saisie de programmes sur plusieurs lignes;
 - (b) l'affichage des messages d'erreurs lexicales comme dans l'exemple suivant : Ligne 5 position 13 : '#' caractère illégal!

 où 13 indique la position du caractère illégal dans la ligne 5 et non pas sa position dans le programme dans son ensemble.

Indication:

On aura besoin pour cela:

- d'utiliser des variables supplémentaires pour contenir les numéros de ligne et de colonne (la position dans la ligne) du dernier caractère lu,
- de modifier légèrement les fonctions :

 $car_suivan($), $erreur_lexicale($) et reculer() pour mettre à jourles nouvelles variables et modifier l'affichage.