Analyseur syntaxique et sémantique

Le but est de transformer le programme qui réalise l'analyse lexicale afin de : (i) prendre en compte l'analyse sémantique et (ii) réaliser la gestion des erreurs.
Pour améliorer l'analyse lexicale, il faut vérifier que les identificateurs utilisés sont bien déclarés. Pour cela, il est nécessaire de mémoriser les identificateurs déclarés pour tester la déclaration ou non des identificateurs utilisés. Cette mémorisation se fait dans une *table des symboles*.

L'analyse sémantique consiste à gérer la table de symbole qui représente l'ensemble des identificateurs utilisés dans un programme. Cette table est désignée par la suite par la variable TABLESYM.

Analyseur syntaxique et sémantique

Dans cette table, on associe à chaque entrée (chaque identificateur) toutes les informations connues sur lui, pour le moment :

sa forme textuelle (son nom) ;

sa classe, à savoir s'il désigne un programme, une constante, ou une variable.

l'adresse de l'identificateur en mémoire

C'est le rôle du compilateur d'*allouer* ces symboles en mémoire. à chaque symbole, le compilateur doit associer un emplacement mémoire dont la taille dépend du type du symbole.

Une manière simple et naturelle de faire est de choisir les adresses au fur et à mesure de l'analyse des déclarations en incrémentant un *offset* qui indique la place occupée par les déclarations précédentes (variable OFFSET).

Analyseur syntaxique et sémantique

En programmation algorithmique la table serait déclarée ainsi :

```
array [TABLEINDEX] of record
                        NOM : Chaine;
                        CLASSE : CLASSES ;
                        ADRESSE : integer
                          end ;
```

Le type CLASSES représente les différentes classes d'identificateur rencontrés : programme, constante, variable.
Le champ ADRESSE représente l'adresse mémoire de la variable ou la constante dans la pile d'exécution MEM qui a été utilisée précedemment

Analyseur syntaxique et sémantique

La table des symboles est manipulée par deux fonctions :
ENTRERSYM
ajoute le token TOKEN dans la table des symboles avec la classe passée en paramètre ;
Elle gére de plus le champ adresse.
Le champ ADRESSE n'est utilisé que pour les variables ; néanmoins, on l'utilisera pour stocker la valeur des constantes ;
CHERCHERSYM
recherche l'identificateur TOKEN dans la table des symboles parmi les classes permises passées en paramètre et retourne l'index de l'entrée correspondante dans la table des symboles, ou s'arrête sur ERREUR ;
Il faut ensuite modifier les procédures écrites précedemment pour réaliser l'analyseur lexicale.
Ces modifications doivent permettre de gérer la table des symboles et de gérer les erreurs.

Analyseur syntaxique et sémantique

```
type
   TABLEINDEX = 1 .. INDEXMAX ;
   CLASSES = (PROGRAMME, CONSTANTE, VARIABLE) ;
   CLASSET = set of CLASSES ;
var
   TABLESYM : array [TABLEINDEX] of record
      NOM : ALFA ;
      CLASSE : CLASSES
   end ;
   DERNIERSYM : TABLEINDEX ; (* indice du dernier symbole entre *)
```

Analyseur syntaxique et sémantique

ENTRERSYM

ajoute le symbole SYM dans la table des symboles avec la classe passée en paramètre ; une version de base peut être la suivante :

```
procedure ENTRERSYM (C:CLASSES) ;
begin
   if DERNIERSYM = INDEXMAX then ERREUR ;
   DERNIERSYM := DERNIERSYM + 1 ;
   with TABLESYM [DERNIERSYM] do
     begin NOM := SYM ; CLASSE := C end
end ;
```

Analyseur syntaxique et sémantique

CHERCHERSYM
recherche l'identificateur SYM dans la table des symboles parmi les classes permises passées en paramètre et retourne l'index de l'entrée correspondante dans la table des symboles, ou s'arrête sur ERREUR ; on donne ici une version de base :

```
procedure CHERCHERSYM (var INDEX:TABLEINDEX ; PERMIS:CLASSET) ;
var FIN : booleen ;
begin
  INDEX := DERNIERSYM ; FIN := false ;
  while not FIN do begin
    if INDEX = 0 then FIN := true
    else if TABLESYM [INDEX].NOM = SYM
      then FIN := true
      else INDEX := INDEX - 1
  end ;
  if INDEX = 0 then ERREUR (* SYM absent *)
  else if not (TABLESYM[INDEX].CLASSE in PERMIS)
    then ERREUR ;  (* SYM pas de bonne classe *)
end ;
```

Analyseur syntaxique et sémantique

Entrée dans la table des symboles

La procédure ENTRERSYM est utilisée lors de la déclaration de symboles par l'intermédiaire d'une procédure TESTE_ET_ENTRE qui vérifie que le prochain token est celui attendu et qui met à jour la table des symboles

```
procedure TESTE_ET_ENTRE (T:TOKENS ; C:CLASSES) ;
begin
  if TOKEN = T then
    begin
      ENTRERSYM (C) ;
      NEXT_TOKEN
    end
  else ERREUR
end ;
```

Analyseur syntaxique et sémantique

Cette fonction TESTE_ET_ENTRE est appelée lors de la déclaration de symboles ; c'est-à-dire dans les fonctions CONSTS, VARS et PROGRAM, par exemple :

```
procedure PROGRAM ;
begin
   TESTE (PROGRAM_TOKEN) ;
   TESTE_ET_ENTRE (ID_TOKEN, PROGRAMME) ;
   TEST (PT_VIRG_TOKEN) ;
   BLOCK ;
   if TOKEN <> POINT_TOKEN then ERREUR
end ;
```

Consultation de la table des symboles

De manière symétrique, une procédure TESTE_ET_CHERCHE est une évolution de la procédure TESTE qui de plus recherche le symbole SYM dans la table des symboles par un appel à CHERCHESYM, l'index du symbole recherché dans la table des symboles est retourné par la variable PLACESYM
var PLACESYM : TABLEINDEX ;

```
procedure TESTE_ET_CHERCHE (T:TOKENS; PERMIS:CLASSET) ;
begin
   if TOKEN = T then begin
      CHERCHERSYM (PLACESYM, PERMIS) ;
      NEXT_TOKEN ;
   else ERREUR ;
end ;
```

Consultation de la table des symboles

Cette fonction est appelée lors de l'utilisation d'identificateur, c'est-à-dire par FACT, AFFEC, et LIRE. On a par exemple :

```
procedure AFFEC ;
begin
   TESTE_ET_CHERCHE (ID_TOKEN, VARIABLE) ;
   TESTE (AFFEC_TOKEN) ;
   EXPR ;
end ;
procedure FACT ;
begin
   if TOKEN = ID_TOKEN
      then TESTE_ET_CHERCHE (ID_TOKEN, [CONSTANTE, VARIABLE])
   else if TOKEN = NUM_TOKEN
      then NEXT_TOKEN
   else begin
      TESTE (PAR_OUV_TOKEN) ;
      EXPR ;
      TESTE (PAR_FER_TOKEN) ;
   end
end ;
```