Génération de code

Nous allons compléter l'analyseur sémantique construit précedemment pour générer le P-Code correspondant au programme analysé.

Une première chose à faire est de décider de l'allocation des variables. Ensuite, chaque reconnaissance d'une règle de la grammaire déclenche la génération de P-Code à l'aide des procédures GENERER1 (génération d'une instruction P-Code sans opérande) et GENERER2 (génération d'une instruction P-Code à un opérande)

Au niveau du langage à compiler (comme dans les autres langages de haut niveau), on ne se préoccupe pas de la gestion de la mémoire ; on manipule celle-ci par l'intermédiaire de symboles.

C'est le rôle du compilateur d'*allouer* ces symboles en mémoire.
A 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).
A la fin des déclarations, il est possible de déterminer l'emplacement mémoire à réserver dans la pile au début de l'exécution du programme (instruction P-Code INT).

Pour chaque symbole, son adresse d'allocation est stockée dans la table des symboles :

```
var
  TABLESYM : array [TABLEINDEX] of record
    NOM : ALFA ;
    CLASSE : CLASSES ;
    ADRESSE : integer
  end ;
  OFFSET : integer ;
```

On modifie la procédure ENTRERSYM pour tenir compte de cette allocation mémoire :

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

Le champ ADRESSE n'est utilisé que pour les variables ;
On peut l'utiliser pour stocker la valeur des constantes ;
On modifiera alors la procédure CONSTS

Génération de début et fin de programme

Une fois l'allocation des données réalisée, il est nécessaire de réserver l'emplacement suffisant dans la pile P-Code. Cette réservation est faite lors de l'analyse d'un BLOCK par la génération d'une instruction P-Code INST :

```
procedure BLOCK ;
begin
   OFFSET := 0 ;
   if TOKEN = CONST_TOKEN then CONSTS ;
   if TOKEN = VAR_TOKEN then VARS ;
   GENERER2 (INT, OFFSET) ;
   INSTS
end ;
```

Génération de début et fin de programme

Lors de la terminaison de l'analyse d'un programme, il est nécessaire de générer une instruction P-Code d'arrêt du programme, HLT :

```
procedure PROGRAM ;
begin
   TESTE (PROGRAM_TOKEN) ;
   TESTE_ET_ENTRE (ID_TOKEN, PROGRAMME) ;
   TEST (PT_VIRG_TOKEN) ;
   BLOCK ;
   GENERER1 (HLT) ;
   if TOKEN <> POINT_TOKEN then ERREUR
end ;
```

Dans un compilateur :

- lorsque l'on termine une procédure d'analyse c'est que l'on a déjà analysé correctement les phrases correspondant aux procédures appelées dans cette procédure.

- Par exemple lors de l'analyse de l'expression a + b, EXPR va appeler TERM deux fois, une fois pour analyser a et une fois pour analyser b ; l'analyse du + se fait au niveau de EXPR. Si les deux appels réussissent (c'est le cas dans notre exemple) et que le + est bien reconnu, EXPR réussit.

Lorsque la génération est dirigée par la syntaxe, on adopte le même raisonnement : lorsqu'une procédure se termine, on considère que la génération des phrases analysées par les procédures appelées est terminée. Il ne reste plus qu'à générer le code pour l'addition, c'est-à-dire simplement l'instruction P-Code ADD.

On commence par le génération des facteurs pour lesquels on laisse sur la pile P-Code une valeur

```
procedure FACT ;
begin
  if TOKEN = ID_TOKEN then begin
    TESTE_ET_CHERCHE (ID_TOKEN, [CONSTANTE, VARIABLE]);
    with TABLESYM [PLACESYM] do
      case CLASSE of
        CONSTANTE : GENERER2 (LDI, ADRESSE) ;
        VARIABLE : begin
          GENERER2 (LDA, ADRESSE) ;
          GENERER1 (LDV)
        end ;
        PROGRAMME : ;
      end
    end
  end ;
  else if TOKEN = NUM_TOKEN then begin
    GENERER2 (LDI, VAL) ;
    NEXT_TOKEN
  end
  else begin
    TESTE (PAR_OUV_TOKEN) ;
    EXPR ;
    TESTE (PAR_FER_TOKEN)
  end
end ;
```

De même, un terme laisse sur la pile P-Code la valeur du terme. Il est nécessaire de mémoriser le token correspondant à l'opération avant l'analyse de l'opérande gauche du terme (variable OP) :

```
procedure TERM ;
var OP : TOKENS ;
begin
  FACT ;
  while TOKEN in [MULT_TOKEN, DIV_TOKEN] do
    begin
      OP := TOKEN : (* memorise l'operation *)
      NEXT_TOKEN ;
      FACT ;
      if OP = MUL_TOKEN
        then GENERER1 (MUL)
        else GENERER1 (DIV)
    end
end ;
```

L'analyse d'une expression laisse aussi une valeur sur la pile P-Code ; le code est similaire à celui de l'analyse des termes :

```
procedure EXPR ;
 var OP : TOKENS ;
 begin
    TERM ;
    while TOKEN in [PLUS_TOKEN, MOINS_TOKEN] do
      begin
        OP := TOKEN ; (* memorise l'operation *)
        NEXT_TOKEN ;
        TERM ;
        if OP = PLUS_TOKEN
          then GENERER1 (ADD)
          else GENERER1 (SUB)
      end
 end ;
```

Génération des conditions

La génération des conditions (procédure COND) est calquée sur celle des expressions.

Génération des instructions simples

Il n'y pas de code à générer lors de l'analyse du bloc d'instructions (INSTS) ou d'une instruction (INST).

On détaillera la génération à réaliser lors de l'analyse d'une instruction d'affectation et des instructions d'entrée/sortie.

Génération d'une affectation

Une affectation A := expression est générée suivant le modèle

LDA <adresse de A>	empile l'adresse de A
<code>	empile la valeur de l'expression
STO	stocke la valeur de l'expression dans A

Génération d'une affectation (suite)

Le P-Code <code> dépose la valeur de expression sur la pile ; il est généré lors de l'analyse de expression par l'appel EXPR. On a donc :

```
procedure AFFEC ;
bcgin
   TESTE_ET_CHERCHE (ID_TOKEN, VARIABLE) ;
   GENERER2 (LDA, TABLESYM [PLACESYM]. ADRESSE) ;
   TESTE (AFFEC_TOKEN) ;
   EXPR ;
   GENERER1 (STO)
end ;
```