- le type de la constante (typc) est soit entier (0), soit chaine (1);
- la valeur de la constante (valent si typc = 0 et valch si typc = 1).

Il faut introduire ces vérifications sémantiques dans la fonction booléenne sans paramètres DECL_CONST. Les parties de programme qui représentent l'analyse sémantique liée à la déclaration des constantes sont indiquées en gras dans la fonction DECL_CONST:

```
function DECL_CONST : boolean;
{ DECL_CONST _  'CONST' 'IDENT' '=' ('ENT' | 'CH') { ',' 'IDENT' '=' ('ENT' | 'CH') } ';' }
var     non_fin: boolean;               { test de fin de boucle }
        nom_constante: string;          { nom d'une constante }

function DEFINIR_CONSTANTE( nom : string, ul : T_UNILEX ) : boolean;
        { ajout d'une constante dans la table des identificateurs }
        var enreg: T_ENREG_IDENT;               { enregistrement de la constante dans la table des
                                                  identificateurs }
        begin
                if ( CHERCHER( nom)  <> 0 ) then
                { s'il existe déjà un identificateur de même nom dans la table des identificateurs }
                        DEFINIR_CONSTANTE := false
                else
                begin
                        enreg.typ := constante;
                        if ( ul = ENT ) then
                        begin
                                enreg.typc := 0{ la constante est de type entier (0) }
                                enreg.val := NOMBRE
                        end
                        else
                        begin
                                enreg.typc := 1{ la constante est de type chaîne (1) }
                                NB_CONST_CHAINE := NB_CONST_CHAINE + 1;
                                VAL_DE_CONST_ CHAINE[ NB_CONST_CHAINE] := CHAINE;
                                enreg.val := NB_CONST_CHAINE
                        end
                        INSERER( enreg );
                        DEFINIR_CONSTANTE := true
                end;
begin
        if (UNILEX = MOTCLE) and (CHAINE = 'CONST') then
        begin
                UNILEX := ANALEX;
                if (UNILEX = IDENT) then
                begin
                        nom_constante := CHAINE;
                        UNILEX := ANALEX;
                        if (UNILEX = EG) then
                        begin
                                if (UNILEX = ENT) or (UNILEX = CH) then
                                begin
                                        if ( DEFINIR_CONSTANTE( nom_constante, UNILEX) ) then
                                        begin
                                           UNILEX := ANALEX;
                                           non_fin := true;
                                           repeat
                                                if ( UNILEX = VIRG ) then
                                                begin
                                                        if (UNILEX = IDENT) then
                                                        begin
                                                                nom_constante := CHAINE;
                                                                UNILEX := ANALEX;
                                                                if (UNILEX = EG) then
                                                                begin
                                                                        if (UNILEX = ENT) or
                                                                          (UNILEX = CH) then
                                                                        begin
                                                                           if ( DEFINIR_CONSTANTE (
                                                                              nom_constante,
                                                                              UNILEX) ) then
                                                                           begin
```

```
                                                    UNILEX := ANALEX;
                                                    non_fin := true;
                                                end
                                                else
                                                    non_fin := false
                                            end
                                            else
                                                non-fin := false
                                        end
                                        else
                                            non_fin := false
                                    end
                                    else
                                        non_fin := false
                                end
                                else
                                    non_fin := false
                            until non_fin = false;
                            if ( UNILEX = PTVIRG ) then
                            begin
                                UNILEX := ANALEX;
                                DECL_CONST := true
                            end
                            else
                                DECL_CONST := false   { erreur syntaxique dans la
                                déclaration des constantes: point virgule attendu }
                        end
                        else
                            DECL_CONST := false    { erreur sémantique dans
                        la déclaration des constantes: identificateur déjà déclaré }
                    end
                    else
                        DECL_CONST := false     { erreur syntaxique dans la
                                                  déclaration des constantes:
                                                  entier ou chaîne attendu }
                end
                else
                    DECL_CONST := false     { erreur syntaxique dans la déclaration des
                                              constantes: symbole = attendu }
            end
            else
                DECL_CONST := false     { erreur syntaxique dans la déclaration des
                                          constantes: identificateur attendu }
        end
        else
            DECL_CONST := false     { erreur syntaxique dans la déclaration des constantes:
                                      mot-clé CONST attendu }
end;
```

La forme des parties de programme ajoutées à la fonction DECL_CONST dépend de la structure de données choisie pour la table des identificateurs et de la définition des fonctions CHERCHER et INSERER. La forme précédente correspond aux conditions suivantes:
- la fonction CHERCHER est une fonction entière qui renvoie:
  0, si le nom n'est pas dans la table des identificateurs,
  l'indice dans la table des identificateurs (ou dans la table de hachage) où se trouve le nom, sinon;
- la fonction INSERER est une fonction entière définie avec un seul paramètre de type T_ENREG_IDENT.

Par contre, si la fonction INSERER admet deux arguments, le nom de l'identificateur et son genre (variable ou constante), et si la table des identificateurs TABLE_IDENT est représentée par un tableau d'éléments du type T_ENREG_IDENT, trié (avec éventuellement une table d'index), la fonction DEFINIR_CONSTANTE devient:

```
function DEFINIR_CONSTANTE( nom : string, ul : T_UNILEX ) : boolean;
    { ajout d'une constante dans la table des identificateurs }
    var indice_insertion : integer;
    begin
        if ( CHERCHER( nom) <> 0 ) then
        { s'il existe déjà un identificateur de même nom dans la table des identificateurs }
            DEFINIR_CONSTANTE := false
        else
        begin
            indice_insertion := INSERER( nom, constante);
            if ( ul = ENT ) then
            begin
                TABLE_IDENT[ indice_insertion].typc := 0;
                TABLE_IDENT[ indice_insertion].val := NOMBRE
            end
            else
            begin
                TABLE_IDENT[ indice_insertion].typc := 1
                NB_CONST_CHAINE := NB_CONST_CHAINE + 1;
                VAL_DE_CONST_ CHAINE[ NB_CONST_CHAINE] := CHAINE;
                TABLE_IDENT[ indice_insertion].val := NB_CONST_CHAINE
            end
            DEFINIR_CONSTANTE := true
        end;
```

Enfin, si la fonction INSERER admet deux arguments, le nom de l'identificateur et son genre (variable ou constante), et si la table des identificateurs TABLE_IDENT est réalisée en utilisant une table de hachage TABLE_IDENT telle que toute insertion est réalisée en début de liste, il faut remplacer, dans la fonction précédente, toutes les occurrences de TABLE_IDENT[indice_insertion].typ par TABLE_IDENT[indice_insertion]^.typ. Remarquons que, dans ce cas, la valeur de indice_insertion est égale à h(nom) où h est la fonction de hachage.

Ajout des variables globales NB_CONST_CHAINE et VAL_DE_CONST_CHAINE

NB_CONST_CHAINE, le nombre de constantes de type chaine reconnues dans le programme source, qui doit être initialisé à 0 dans la procédure INITIALISER.
VAL_DE_CONST_CHAINE, le tableau des valeurs des constantes de type chaine.

Fonction associée au non terminal DECL_VAR de la grammaire G_0

Lors de la déclaration d'une variable, il faut effectuer la vérification sémantique suivante:
le nom de la variable ne doit pas être un nom d'identificateur déjà présent dans la table des identificateurs.
Si ce nom est déjà défini, alors il faut produire l'erreur « identificateur déjà déclaré ». Sinon, ce nom n'a pas encore été défini dans la table des identificateurs, il faut ajouter la variable dans la table des identificateurs en remplissant les différents champs spécifiques à cette variable:
- le nom de la variable (nom) ;
- le type de l'identificateur (typ) est: variable;
- le type de la variable (typc) est toujours entier (0);
- l'adresse de la variable est son indice dans la zone mémoire des variables globales MEM_VAR (cf. §4 p.10).

Chaque fois, qu'une variable est ajoutée dans la table des symboles, il faut incrémenter DERNIERE_ADRESSE_VAR_GLOB de 1, et l'adresse de la variable est égale à la nouvelle valeur de DERNIERE_ADRESSE_VAR_GLOB. L'adresse de la première variable globale est 0.
Il faut introduire ces vérifications sémantiques dans la fonction booléenne sans paramètres DECL_VAR.

Remarques:
- Une variable ou une constante n'est ajoutée à la table des identificateurs qu'à la fin de sa définition. Ceci permet d'éviter des définitions erronnées, comme par exemple CONST xy = xy;
- Lorsqu'on ajoute une constante ou une variable à la table des identificateurs, il faut penser à vérifier au préalable que la table n'est pas pleine, dans le cas où la table est gérée comme un tableau de taille fixe.

Ajout de la variable globale DERNIERE_ADRESSE_VAR_GLOB

DERNIERE_ADRESSE_VAR_GLOB contient la dernière adresse associée à une variable globale. Elle est initialisée à -1 dans la procédure INITIALISER. Après la lecture de toutes les déclarations de variables du programme source mini-Pascal, sa valeur est égale au nombre de variables déclarées moins un.

Fonctions associées aux non terminaux AFFECTATION, TERME, LECTURE **de la grammaire G_0**

- l'identificateur en partie gauche d'une affectation doit être une variable qui, de plus, doit avoir été déclarée, il faut le vérifier dans la fonction AFFECTATION.
- l'identificateur intervenant dans une expression arithmétique doit avoir été déclaré, il faut le vérifier dans la fonction TERME.
- dans l'instruction de lecture, les identificateurs ne peuvent être que des variables et ces variables doivent avoir été déclarées, il faut le vérifier dans la fonction LECTURE.

Vérification de types dans la fonction associée au non terminal TERME **de la grammaire G_0**

La deuxième catégorie d'erreurs sémantiques résulte de la vérification de types. La règle de compatibilité des types en mini-Pascal est simple: Deux objets sont de type compatibles s'ils sont déclarés avec le même type (valeur du champ typc ou typv dans la table des symboles).
Les vérifications de type doivent se faire à différents endroits:

- les parties gauche et droite dans une affectation sont toujours de type compatibles en mini-Pascal puisque les variables sont toutes de type entier et les expressions arithmétiques sont entières.
- les opérandes de gauche et de droite d'un opérateur doivent être de type compatibles; par conséquent, il faut vérifier dans la fonction TERME que les constantes qui interviennent dans les expressions arithmétiques du mini-Pascal sont de type entier.

Le programme principal

Le programme principal de l'analyseur syntaxique reste inchangé. Il faut tester l'analyse sémantique. Pour cela, on pourra faire afficher les informations de la table des identificateurs, chaque fois qu'un identificateur est ajouté. Il faudra ensuite construire et utiliser quatre programmes sources en mini-Pascal qui produisent chacun une erreur sémantique différente parmi celles recensées dans les fonctions AFFECTATION, TERME et LECTURE.

3. Messages d'erreur

Quelle que soit l'erreur syntaxique ou sémantique détectée ('erreur syntaxique dans la déclaration des constantes: mot-clé CONST attendu', ou 'erreur sémantique dans la déclaration des constantes: identificateur déjà déclaré', etc.), il est possible d'afficher le même message: 'ligne x, erreur syntaxique ou sémantique'. Cet affichage est effectué par l'appel ERREUR(3) dans la procédure ANASYNT. ERREUR(3) provoque alors l'exécution du code suivant:

writeln('Ligne ', NUM_LIGNE, ': erreur syntaxique'); halt;

Il est possible d'afficher un message plus explicite en utilisant une variable globale de type chaîne, MESSAGE_ERREUR, qui est affectée au moment où une erreur syntaxique ou sémantique est détectée. Par exemple, lorsque l'erreur « mot-clé CONST attendu » est détectée, MESSAGE_ERREUR prend la valeur 'erreur syntaxique dans la déclaration des constantes: mot-clé CONST attendu'. L'appel ERREUR(3) doit alors provoquer l'exécution du code suivant: writeln('Ligne ', NUM_LIGNE, ': ', MESSAGE_ERREUR); halt;

4. Génération de code et interprétation

Notre objectif n'est pas d'écrire un compilateur générant du code pour une machine X ou Y. Il faudrait, pour cela, une connaissance parfaite du jeu d'instructions de la machine cible avec tous ses modes d'adressage, etc. Le code que nous allons générer avec notre compilateur mini-Pascal est un code pour une **machine virtuelle**, décrite ci-après. Pour pouvoir exécuter les programmes mini-Pascal sur un ordinateur quelconque, il suffit alors d'écrire un **interpréteur** pour les instructions de la machine virtuelle.

⇨ Organisation de la mémoire de la machine virtuelle

La mémoire centrale de la machine virtuelle est structurée en trois parties:

- le bas de la mémoire, à partir de l'adresse 0, est une partie de taille fixe durant l'exécution du programme, réservée aux variables globales du programme;
- la partie suivante est une zone de mémoire de taille fixe durant l'exécution du programme réservée au code du programme;
- au-dessus, le reste de la mémoire est utilisé comme pile d'exécution; cet espace mémoire est alloué dynamiquement au fur et à mesure des besoins lors de l'exécution du programme.