Compte Rendu TP compilation Analyse Sémantique

Réalisé par
Meriem Ben Chaaben & Fares el kahla

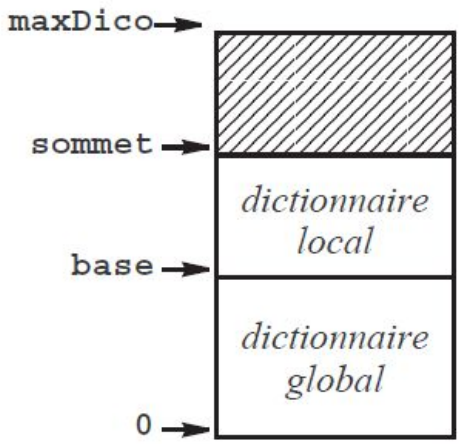
On a créé un fichier semantic.h pour la déclaration des structures , un fichier semantic.c qui contient toute la logique de l'analyse sémantique .
NB : En testant , la dernière instruction avant le End. ne demande pas **" ; "** à la fin . Sinon on aura une erreur syntaxique.

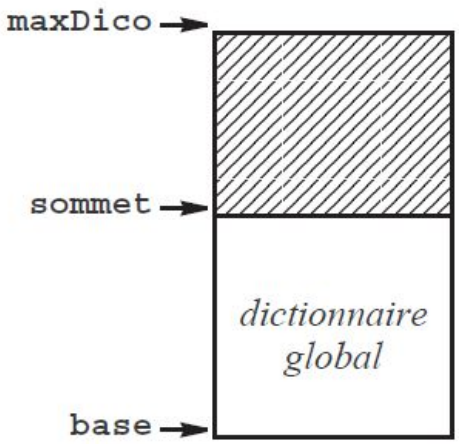
1. Structures utilisées :

On a suivi la structure proposée dans le document **polyCompile.**

a. Dictionnaire (tables de symboles):

Il s'agit d'un tableau où les identificateurs sont placés dans l'ordre ou leurs déclarations ont été trouvées dans le code à compiler.
ENTREE_DICO , une structure qui dispose d'un élément nommé type , qui décrit à la fois le type de l'élément ajouté dans la table de symboles et qui s'occupera du chaînage des éléments(variables , procédures , fonctions , arguments ...)
On dispose de deux variables globales , **Sommet** et **Base** qui servent à vérifier les scopes , on parle en effet de Dictionnaire global & dictionnaire local.





2. Contraintes :

a. Vérifier la redéfinition des variables déjà déclarées
Il suffit de parcourir le dictionnaire des données en considérant les scopes.
VerifAjout

b. Vérifier l'appel des procédures avec les bons arguments
En créant l'identificateur d'une Procédure on prépare aussi comme attribut une liste de paramètres (Queue) .Ainsi si on appelle cette procédure , on crée aussi une liste avec les arguments passés et ensuite on compare les deux listes en terme de nombre d'arguments , leurs types …

verifMethodCall

c. **Vérifier qu'une variable utilisée est bien déclarée.**
   Il suffit de parcourir le dictionnaire des données en considérant les scopes.

d. **Vérifier que les variables déclarées sont bien initialisées.**
   On dispose d'un attribut initialised pour chaque élément de notre tableau. Cet attribut est mis à 0 à l'ajout et si on dispose d'affectation il sera mis à 1.
   A la fin du parcours de grammaire sémantiquement , on appelle la methode de verifInitialised

e. **Vérifier qu'une variable déclarée est bien utilisée.**
   On dispose d'un attribut initialised pour chaque élément de notre tableau. Cet attribut est mis à 0 à l'ajout et si on dispose d'affectation il sera mis à 1.
   verifUsed
   On Note qu'on a considéré cette contrainte comme Warning et non pas une erreur

3. Pour aller plus loin :

a. On a aussi traité les fonctions (non seulement les procédures) , on a considéré donc le type de retour et on l'a vérifié lors de l'appel.
b. Il est possible de faire un **surcharge** de fonctions ou de procédures c'est à dire , deux méthodes ayant le même nom mais avec des arguments différents seront acceptés.
c. On dispose comme type primitif le type **Boolean** aussi.
d. Pour vérifier la compatibilité des arguments utilisés lors d'un appel d'une fonction ou d'une procédure ou lors d'une affectation et vu qu'un argument peut être le résultat d'une expression quelconque on a préciser pour chaque opérateur utilisé (mod , or , and ,div , …) le type de résultat retourné exacte . par exemple mod donne comme type de retour int ...
e. On affiche lors de la compilation les identificateurs déclarés et les variables/ paramètres utilisés en spécifiant leurs types.
f. Dans le cas d'une erreur sémantique on ne fait pas arrêter l'analyse , on affiche un message explicite , on mentionne la ligne ou il y a un problème et le numéro de l'erreur => l'analyse poursuit jusqu'à la fin pour permettre d'afficher éventuellement plusieurs erreurs a la fois ainsi donner la possibilité de les fixer une fois pour toute .
g. La couleur de l'affichage diffère , si il s'agit d'un warning (Jaune) , une erreur (rouge) ou si tout va bien (noir).

4. Comment utiliser le produit :

a. Flex pascal .lex
b. Bison -d pascal.y
c. Gcc -o pascal3.exe pascal.tab.c lex.yy.c
d. Lancer le fichier interface.py
    i. Compiler un code source à partir d'un fichier en donnant le path (avec browser ou manuellement)

ii. Compiler un code source en le tapant dans “text area”

5. Codes testé dans la démo :

1- Test commentaire :

```
/* ============================================================
  CECI EST TEST DE COMMENTAIRE

  ============================================================ */
program TP3sematinc ;
var x,y: integer;
begin
  x := 3;
  y := 3*2;
  write(x,y,45+2) /*mechant programme n'est pas ? */
end .
```

2- Vérifier la redéfinition des variables déjà déclarées

a- 2 variables memes types : error

```
program TP3sematinc ;
        var x,x: integer;
begin
end .
```

b- 2 variables types differents : error

```
program TP3sematinc ;
        var x: integer;
        var x: double ;
begin
end .
```

c- variable et procedure : accepted

```
program TP3sematinc ;
        var x: integer;
procedure x() ;
        begin
        end ;
begin
end .
```

d- 2 procédures/fonctions meme signature : error

```
program TP3sematinc ;
procedure x() ;
        begin
        end ;
```

```
procedure x() ;
        begin
        end ;
begin
end .
```

d- 2 overloading d'une procedure/fonction meme signature : accepted

```
program TP3sematinc ;
procedure x() ;
        begin
        end ;
procedure x(a : integer) ;
        begin
        end ;
begin
end .
```

e- 2 variables meme scope : error

```
program TP3sematinc ;
procedure x(a: array [1 .. 10 ] of integer) ;
        var a: integer;
        begin
        end ;
begin
end .
```

f- 2 variables scope different : accepted

```
program TP3sematinc ;
        var a: integer;
function x(a:integer):double ;
        begin
        end ;
begin
end .
```

3-Vérifier l'appel des procédures avec les bons arguments

a- sans parametres : accepted

```
program TP3sematinc ;
        var x: integer;
procedure p() ;
        begin
        end ;
begin
        p()
end .
```

b- avec parametres et on passe rien : error

```
program TP3sematinc ;
        var x: integer;
procedure p(a : integer) ;
        begin
        end ;
begin
        p()
end .
```

c- sans parametres et on passe un parametre : error

```
program TP3sematinc ;
        var x: integer;
procedure p() ;
        begin
        end ;
begin
        p(x)
end .
```

d- avec parametre et on passe un parametre de type errorné : error

```
program TP3sematinc ;
        var x: integer;
procedure p(a: integer) ;
        begin
        end ;
begin
        p(4.2)
end .
```

e- bonne affectation du type de retour : accepted

```
program TP3sematinc ;
        var x: integer;
function f(a: integer):integer ;
        begin
        end ;
begin
        x := f(2)
end .
```

f- mauvaise affectation du type de retour : error

```
program TP3sematinc ;
        var x: integer;
function f(a: integer):double ;
        begin
        end ;
```

```
begin
        x := f(2)
end .
```

g- mauvaise affectation du type de retour mais avec subttle casting int to double: accepted

```
program TP3sematinc ;
        var x: double;
function f(a: integer):integer ;
        begin
        end ;
begin
        x := f(2)
end .
```

h- manipulation des booleans : error

```
program TP3sematinc ;
        var a: boolean;
begin
        a := 2.5
end .
```

i- manipulation des booleans : accepted

```
program TP3sematinc ;
        var a: boolean;
begin
        a := True ;
        a := (a and (2 > 3 ))
end .
```

j- manipulation des strings : error

```
program TP3sematinc ;
begin
        writeln("string1" + 4 )  ;
        writeln("string1" *3 )
end .
```

k- manipulation des strings : accepted

```
program TP3sematinc ;
begin
        writeln("string1" + "string2")
end .
```

l- if then et condition de type boolean : accepted

```
program TP3sematinc ;
begin
        if True then
```

```
                writeln("accepted")
end .
```

m- if then et condition de type boolean : error

```
program TP3sematinc ;
begin
        if 5.7 then
                writeln("error")
end .
```

4-Vérifier qu'une variable utilisée est bien déclarée.

```
program TP3sematinc ;
begin
        x:= 1
end .
```

5-Vérifier que les variables déclarées sont bien initialisées.

```
program TP3sematinc ;
        var x,y : integer ;
begin
        y:= x+ 2 ;
        writeln(x, y)
end .
```

6-Vérifier qu'une variable déclarée est bien utilisée.

```
program TP3sematinc ;
        var x,y : integer ;
procedure p() ;
        begin
        end ;
begin
        x:=1 ;
        y:= 2 ;
        p();
        writeln(x, y)
end .
```