Bréviaire de EXALGO - Module ASD

La définition (due à J.G. Penaud) de ce mini langage permet de fixer les quelques règles élémentaires permettant d'écrire des algorithmes en s'affranchissant l'implémentation.

Generalités

Le langage EXALGO est composé de chaînes de caractères alphanumériques, de signes opératoires, de mot-clés réservés, et de signes de ponctuation : **":=, ;,(,)**, **début** , **fin**, **//**. Les marqueurs de fin, **début** et **fin** peuvent être remplacés par **{** et **}** lorsqu'il y a encombrement.

Structure de l'algorithme

Voici un exemple d'algorithme EXALGO syntaxiquement correct.

```
programme  NomAlgorithme;
    // partie déclarations
    début
      // partie instruction
      // ceci est un 3 ème commentaire
    fin.
```

Type

- Types prédéfinis : **entier**,**car**,**booléen** ,**réél**
- Définition de type :

```
type NomDeType= TypePrédéfini;
```

- Définition d'un tableau d'entiers :

```
typeNomDeType = tableau 1..limite deTypePrédéfini;
```

Variables

```
var NomDeVariable: TypePrédéfini;
```

Expressions

Consituées à l'aide de variables déjà déclarées, de parenthèses et d'opérateurs du (des) type(s) des variables concernées.

Instructions simples

- affectation :

```
NomDeVariable:=ExressionDuTypeDeLavariable;
```

- sortie de calcul :**exit**, **retourne()**

Structure de contrôle

- Bloc d'instruction :

```
instruction1
instruction2
.............
```

- Alternative:

```
si ExpressionBooléenne alors
    BlocInstruction1
sinon
    BlocInstruction2
finsi;
```

- Alternative multiple:

```
 selon que
   cas  cas1 : BlocInstruction1
   cas  cas2 : BlocInstruction2
      ............
   autrement :  BlocInstruction
 finselonque
```

- Répétition : **exit** permet d'arrêter la répétition
  - le bloc d'instruction peut ne pas être éxécuté

```
tant queExpressionBooléenne faire
    BlocInstruction
fintantque;
```

  - le bloc d'instruction peut ne pas être éxécuté et il y a une variable indicatrice

```
pour VariableIndicatrice
  allant de  ValeurInitiale à ValeurFinale
  par pas de ValeurPas faire
      BlocInstruction
finpour;
```

  - le bloc d'instruction est éxécuté au moins une fois

```
répéter
    BlocInstruction
jusqu'à ExpressionBooléenne finrépéter;
```

Procédures et fonctions

Une fonction retourne une valeur par l'instruction simple. Une fonction s'utilise dans le calcul d'une expression. Une procédure s'utilise comme une instruction simple. **retourne**(*Expression*).

- Procédures

```
procédure NomDeProcédure (ListeParamètres);
    //déclarations des variables locales autres que les paramètres
    début
    // partie instruction
    fin
finProcédure;
```

- Fonctions

```
fonction NomDeFonction (ListeParamètres):Typerésultat;
    //déclarations des variables locales autres que les paramètres
    début
    // partie instruction qui contient l'appel à retourne()
    fin
finFonction
```

- Paramètres
  Les paramètres sont passés
  - par référence **ref**,
  - par valeur **val**,

  On écrit :

```
   val  ListeVariable:NomDeType
ou
   ref  ListeVariable:NomDeType
```

Types

- Type enregistrement
  Un type enregistrement est constitué à partir de types de base ou d'autres types déclarés.

  ```
  type NomDeType: enreg
                       champ1:NomDeType1
                       champ2:NomDeType2
                         ...........................
                   finenreg
  ```

  Après la déclaration

  ```
  var E:NomDeTypeEnregistrement
  ```

  on accède au différents champs par le nom de la variable suivi d'un **point** suivi du nom de champ (*E.champ1*)

- Type pointeur
  Si *O* est un objet de type *T*, l'objet es accédé par *O*^. Si on déclare :

  ```
  var P:^NomDeType
  ```

  alors on peut obtenir un objet accessible par **allouer**(*P*). Lorsqu'on n'utilise plus l'objet, il faut libérer l'espace qu'il utilise par **desallouer**(*P*).