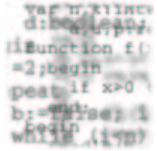

TurboPascal - Prépa HEC Ipecom
Struct. - Vars. - Tabs. - Opés. - Ent/sort.

notes de cours - vendredi 15 mars

1 Structure d'un programme

La structure générale d'un programme est rigoureusement codifiée, en 3 parties: l'en-tête, la déclaration et le corps.

1.1 En-tête

L'en-tête consiste en la définition du nom du programme, sous la forme

```
program NomDuProgramme;
```

`program` est un mot-clé, et `;` signale la fin de l'instruction.
Le nom du programme, comme tous les noms utilisés, doit vérifier plusieurs conditions :

- Il est composé de chiffres, de lettres non accentuées, et du signe souligné [`0-9`,`a-z`,`A-Z`,`_`].
- Il ne débute pas par un chiffre.
- Il est formé de moins de 64 caractères.
- Les minuscules et majuscules sont indifférenciées.

1.2 Déclaration

Elle a elle aussi une syntaxe stricte; on établit la liste des objets qui seront utilisés dans le programme, catégorie par catégorie.

- Après le mot-clé `var` vient une liste des variables qui seront utilisées dans le programme, ainsi que leur type. Pour déclarer une variable `N` de type `integer`, on écrira

```
var N:integer;
```

- Après le mot-clé `const` vient une liste de variables pour lesquelles une valeur est fixée dans tout le programme. Pour associer à la constante `a` la valeur `3`, on écrira

```
const a=3;
```

Ici aissi, on a des mots-clé, et le `;` termine les instructions.
On pourra aussi déclarer des `type`, `function` et `procedure`, comme on le verra plus tard. Les nuances de syntaxe [`:` ou `=`] sont impératives. D'autre part, l'ordre des différentes déclarations est indifférent; les déclarations peuvent également être regroupées, la seule contrainte étant de tout déclarer avant le corps du programme.
On peut donc avoir une déclaration de la forme

```
const a=1;
      b=2;
      c=3;
var N1,N2,N3:integer;
    R1,R2:real;
```

1.3 Corps

Le corps contient l'essentiel du programme, c'est-à-dire la liste des actions à effectuer; pour l'instant, on note juste qu'il doit débuter par le mot-clé `begin` [sans point-virgule], et terminer par `end.` [le `.` est impératif].

1.4 Remarques

- L'usage de majuscules ou de minuscules est indifférent dans tout le programme; cela signifie aussi que pour le programme, l'objet `A` et l'objet `a` sont les mêmes...
- Les espaces et passages à la ligne peuvent être librement insérés, afin d'améliorer la lisibilité.
- On peut également ajouter des commentaires, non lus par le programme entre accolades { } ou entre parenthèses-étoiles `(* *)`.

2 Variables

Les objets que l'on crée et modifie sont appelés des variables, c'est-à-dire des emplacements de mémoire de l'ordinateur qui sont à notre disposition pour stocker temporairement des nombres; il est nécessaire d'indiquer le type de donnée stockée.

Les deux types les plus utilisés sont:

- `integer`, qui correspond à un nombre entier entre -32768 et 32767 [pour des valeurs au-delà, les calculs seront faux].
- `real`, qui correspond en fait à un nombre à virgule, avec une 11 chiffres significatifs , entre 2.9E-39 et 1.7E38.

On a également les types secondaires suivants:

- `shortint`, `byte`, `word`, `longint`, pour des entiers de grandeur différente [-127 128, 0 255, 0 65535, -2147483648 2147483647].
- `single`, `double`, `extended`, pour des nombres 'à virgule' de grandeur différente [1.5E-45 3.4E+38, 5E-324 1.7E+308, 1.9E-4951 1.1E4932].
- `boolean`, pour des tests logiques; un booléen vaudra donc `TRUE` ou `FALSE`, et on pourra effectuer des opérations logiques [`AND`, `OR`, ...].
- `char`, pour un caratère alphanumérique [déclaré par `var C:char;` et défini par `C:='a';`, par exemple], et `string`, pour une chaîne de caractères [déclarée par `var S:string[20];` pour une chaîne de 20 caractères et définie par `S:='gloubiboulga';`, par exemple].

Comme tout objet défini et utilisé, leur nom doit respecter les conditions énoncées plus haut.

3 Tableaux

Les tableaux permettent de définir des variables groupées, et repérées par un indice. Un tableau de 20 nombres entiers, repérés par les indices 1 à 20, est défini par

```
var T:array[1..20] of integer;
```

et un tableau de 8 booléens, repérés par les indices 11 à 18, est défini par

```
var T2:array[11..18] of boolean;
```

On peut alors obtenir le 5ième élément du tableau `T` par `T[5]` [on ne peut pas traiter plusieurs éléments à la fois, comme par `T2:=T1[3..5];`, par exemple].

Il est aussi possible de définir des tableaux bi-dimensionnels [correspondant à des matrices], de deux manières:

- `var TT:array[1..5,1..10] of real;` par exemple définit un tableau à 5 lignes et 10 colonnes de nombres réels. L'élément de la 3ième ligne et 6ième colonne est alors repéré par `TT[3,6]`.
- `var TT:array[1..5] of array[1..10] of real;` par exemple définit un tableau à 5 éléments dont chaque élément est un tableau de 10 nombres réels. On a ici aussi 5 lignes et 10 colonnes, mais l'élément de la 3ième ligne et 6ième colonne est alors repéré par `TT[3][6]`, car `TT[3]` désigne le 3ième élément de `TT`, c'est-à dire la ligne des 10 nombres réels.

Enfin, on utilise souvent dans le contexte des tableaux des types prédéfinis [ceux-ci peuvent également être utilisés dans d'autres cadres]; il s'agit d'une sorte de raccourci, indiqué dans la déclaration du programme, qui définit un type précis de variable pour lequel on choisit un nom. Par exemple, on pourra avoir

```
const N=10;
type vecteur=array[1..N] of real;
     matricecarree=array[1..N] of vecteur;
var A,B,C:matricecarree;
    X,Y:vecteur;
```

4 Operations et fonctions simples

Les variables déclarées, il faut les manipuler, mais surtout les remplir! On appelle cela l'affectation des variables. Au début, le contenu des variables déclarées est aléatoire, pas du tout mis à zéro. Il faut alors souvent définir ce que contiennent les variables.

Pour mettre la valeur `3` dans `nb_entier`, et `Pi` [valeur prédéfinie] dans `nb_reel`, on écrira

```
nb_entier := 3; nb_reel := Pi;
```

Le `:=` est très important!! On n'écrit pas une égalité, mais une affectation [mettre une valeur dans une case mémoire...]

On peut ensuite modifier les variables à l'aide d'opérateurs classiques [`+ - * /`, avec les parenthèses `()`], et quelques fonctions usuelles.

- Fonctions traitant les `integer` et renvoyant des `integer`: `div`, `mod` [division entière et son reste : `17 div 3` donne `6` et `17 mod 3` donne `1`], `abs()`, `sqr()` [valeur absolue et carré].
- Fonctions traitant les `integer` et les `real` et renvoyant des `real`: `abs()`, `sqr()`, `sqrt()` [racine carrée], `sin()`, `cos()`, `ln()`, `exp()` [fonctions usuelles].
- Fonctions traitant les `real` et renvoyant des `integer` : `round()` [arrondi], `trunc()` [troncature].
- Fonctions traitant les `real` et renvoyant des `real` : `int()` [partie entière], `frac()` [partie fractionnaire].

Ces fonctions s'utilisent sous la forme `A:=cos(B); C:=div(13,5);...`
On notera que la fonction 'exposant' n'existe pas.

5 Entrées/sorties

Enfin, un minimum d'interactivité est nécessaire. Pour cela, l'essentiel est constitué de deux fonctions [commandes]:

- `write('Un texte')` écrit le texte [qui peut contenir des caractères alphanumériques et des espaces] à l'écran.
- `write('Un texte',a,b,'Autre texte')` écrit le premier texte, puis le contenu des variables `a` puis `b`, et le second texte.
- `read(a)` afiche le curseur à l'écran, et attend que l'utilisateur saisisse une valeur, qui est affectée dans la variable `a` lorsque l'utilisateur valide par RETURN.
- `read(a,b)` afiche le curseur à l'écran, et attend que l'utilisateur saisisse deux valeur, séparées par un espace, qui sont affectées dans la variables lorsque l'utilisateur valide par RETURN.
- La commande `writeln()` agit comme `write()`, mais passe ensuite à la ligne suivante.
- La commande `readln()` agit exactement, dans le cadre de nos programmes, comme `read()`.
- Si le programme contient l'instruction `readln;` [sans parenthèses], alors l'exécution de celui-ci s'arrètera à cette instruction, et attendra que l'utilisateur tape sur RETURN; cela permet de faire une pause.