Correction TP2 - Types de variables

Exercice 02.1

Dans le programme de cet exercice, il se passe les choses suivantes :
- On donne à l'ordinateur les premières valeurs prises par les variables `x` et `y`. Ce sont des réels, notons-les x_0 et y_0.
- Première modification : on met la valeur $x_0 + y_0$ dans la variable `x`.
- Seconde modification : on met dans `x` la valeur contenue dans `x` moins la valeur contenue dans `y`, c'est-à-dire $(x_0 + y_0) - x_0 = y_0$.
- A la fin, on a donc que la variable `x` a la valeur y_0 et `y` a la valeur x_0.

Si on échange les deux dernières affectations, on n'obtient pas le même résultat :
- Première modification : `x` contient $x_0 + y_0$ et `y` contient y_0.
- Seconde modification : `x` contient $(x_0 + y_0) - y_0 = x_0$, `y` contient y_0.
- A la fin, `x` contient x_0 et `y` contient $x_0 - y_0$.

Exercice 02.2

Dire que T secondes équivalent à H heure(s), M minute(s) et S seconde(s), c'est dire que :

$$T = H \times 3600 + M \times 60 + S,$$

avec $S < 60$ et $60 \times M + S < 3600$.
H est donc le quotient de la division euclidienne de T par 3600, et si on note R le reste, M est le quotient de la division de R par 60, et S est le reste. On en déduit le programme suivant :

```
PROGRAM conversion_temps ;
    VAR T,H,R,M,S :INTEGER ;
BEGIN
    WRITE('Donner le temps en secondes :') ;
    READLN(T) ;
    H :=T DIV 3600 ;
    R :=T MOD 3600 ;
    M :=R DIV 60 ;
    S :=R MOD 60 ;
    WRITELN(H,' heure(s), ',M,' minute(s), ',S,' seconde(s).') ;
    READLN ;
END.
```

On n'est pas obligé de créer 5 variables informatiques pour écrire le programme, on peut aussi écrire :

```
PROGRAM conversion_temps ;
    VAR T :INTEGER ;
BEGIN
    WRITE('Donner le temps en secondes :') ;
    READLN(T) ;
    WRITELN(T DIV 3600,' heure(s), ',(T MOD 3600) DIV 60,' minute(s), ',
        (T MOD 3600) MOD 60,' seconde(s).') ;
    READLN ;
END.
```

Exercice 02.3

Rien à signaler.

Exercice 02.4

1. Ici, il faut corriger les expressions que l'on veut mettre dans la variable `d`. En effet, si `b` et `c` contiennent des entiers, la phrase mathématique "`b and c`" ne veut rien dire !

   On souhaite seulement savoir si `a` est plus petit que `b` et plus petit que `c`. Il faut donc tester si "`a < b`" ET "`a < c`". On doit donc changer la ligne d'affectation de `d` par :

   ```
   d := ((a<b) and (a<c)) ;
   ```

2. Ici encore, il faut corriger les expressions que l'on veut mettre dans la variable `d`. En effet, si `b` contient un entier, "une phrase mathématique OU b" ne veut rien dire !

   On souhaite seulement savoir si `a` est plus petit que `b` ou égal à `b`. Il faut donc tester si "`a < b`" OU "`a = b`". On doit donc changer la ligne d'affectation de `d` par :

   ```
   d := ((a<b) or (a=b)) ;
   ```

   Remarquons qu'on aurait pu laisser uniquement l'expression

   ```
   d := (a<=b) ;
   ```

   qui contenait déjà entièrement ce qu'on voulait vérifier.

Exercice 02.5

1. $$227 = 1 + 2 \times 113 = 1 + 2(1 + 2 \times 56) = 1 + 2 + 2^2 \times 56$$

   Comme $56 = 8 \times 7 = 2^3(1 + 2 + 4) = 2^3 + 2^4 + 2^5$, on a donc

   $$227 = 1 + 2 + 2^5 + 2^6 + 2^7,$$

   et son code binaire est 110001110000000.

2. $$1008 = 2 \times 504 = 2^2 \times 252 = 2^3 \times 126 = 2^4 \times 63$$

   Comme $63 = 1 + 2 \times 31 = 1 + 2(1 + 2 \times 15) = 1 + 2 + 2^2 + 2^3 \times 7 = 1 + 2 + 2^2 + 2^3 + 2^4 + 2^5$, on a donc

   $$1008 = 2^4 + 2^5 + 2^6 + 2^7 + 2^8 + 2^9,$$

   et son code binaire est 000011111100000.