FONCTIONS BOOLÉENNES ET FONCTIONS RÉCURSIVES

ISN - Chapitre 7

Table des matières

I Les fonctions booléennes

I 1 L'expression des fonctions booléennes

Ces fonctions associent à un ou plusieurs booléen(s) un unique booléen. Ces fonctions sont définies sur un ensemble fini et seront le plus souvent présentée à l'aide de table appelées « table de vérité ».

Fonction « non »		**Fonction « et »**			**Fonction « ou »**		
x	$non(x)$	x	y	x et y	x	y	x ou y
0	1	0	0	0	0	0	0
1	0	0	1	0	0	1	1
		1	0	0	1	0	1
		1	1	1	1	1	1

Convention :
0 est lu comme « faux » et 1 est lu comme « vrai ».

Remarque :
La fonction « ou » est **inclusive** ou **exclusive** :
« ou » inclusif : « Je viendrai, qu'il pleuve ou qu'il vente ».
« ou » exclusif : « Tu vas à la mer ou tu vas à la montagne. »
La fonction « ou exclusif » est notée « oux » (*xor* en anglais).

Fonction « oux »		
x	y	x oux y
0	0	0
1	0	1
0	1	1
1	1	0

I 2 L'expression des fonctions booléennes avec les fonctions « non », « et » et « ou »

On peut exprimer de manière symbolique toutes les fonctions booléennes avec les seules fonction « non », « et » et « ou ».

Fonction multiplexeur, mux :
Cette fonction de $\{0\,;1\}^3$ dans $\{0\,;1\}$ est telle que :
• Si x vaut 0, alors $mux(x,y,z)$ vaut y.
• Si x vaut 1, alors $mux(x,y,z)$ vaut z.

Définir la table de vérité de la fonction mux :

x	y	z	$mux(x,y,z)$
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Exercice :
Montrer que $mux(x,y,z) = (non(x)\ \ et\ \ y)\ \ ou\ \ (x\ \ et\ \ z)$.

Exercice :
- Déterminer l'expression de la fonction identité $i(x)$.
- Déterminer l'expression de la fonction constante égale à 0 notée $h(x)$, à l'aide de la « non », « et » et « ou » (On pourra commencer par faire une table de vérité)
- Déterminer l'expression de la fonction « oux ».
- Déterminer l'expression de la fonction « si et seulement si » définie par la table de vérité suivante :

Fonction « ssi »		
x	y	x ssi y
0	0	1
0	1	0
1	0	1
1	1	1

I 3 L'expression des fonctions booléennes sans la fonction « et »

Il est possible de se passer de la fonction « et », toutes les fonctions booléennes peuvent s'exprimer avec les fonctions « non » et « ou ».

Pour cela, il suffit de montrer que la fonction « et » elle-même peut s'exprimer ainsi. Cette fonction s'exprime de la manière suivante :

$$x \text{ et } y = non(non(x) \text{ ou } non(y))$$

Exercice :
Montrer que x ou $y = non(non(x)$ et $non(y))$.

II Les fonctions récursives

II 1 Des fonctions qui appellent d'autres fonctions

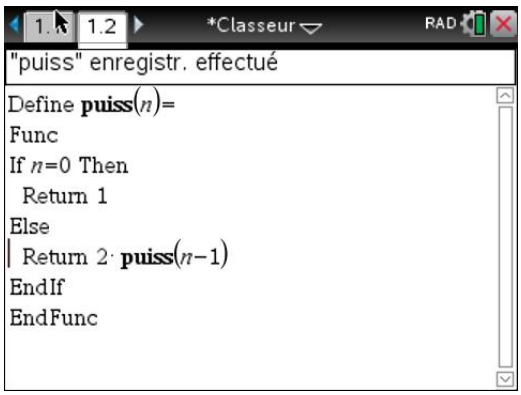
Les fonctions permettent d'isoler des instructions du programme principal.
Intérêts : Alléger le programme principal, améliorer sa lecture. / Optimiser le codage pour éviter de répéter de très nombreuses lignes quasi-identiques.

Pouvez-vous donner des exemples de fonctions appelant d'autres fonctions ?

II 2 Des fonctions qui s'appellent elles-mêmes

Il est même possible d'aller plus loin et d'appeler une fonction f, non depuis le corps d'une fonction g, mais depuis le corps de la fonction f elle-même.

Exemple en TI-Basic :
Codons une fonction qui prend en entrée un entier n et renvoie la valeur de 2^n :

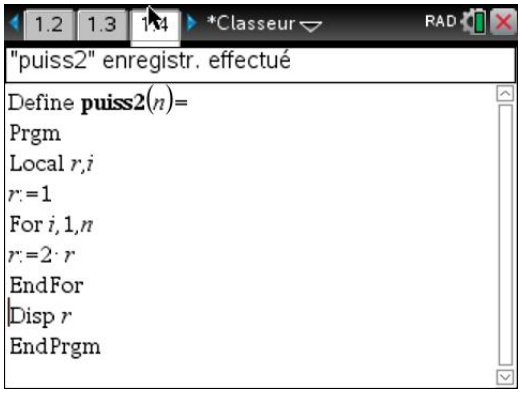


Attention : Il faut définir une *fonction*, et non un *programme*.

Définition

Une fonction qui s'appelle elle-même est appelée une **fonction récursive**.

Remarques :
- Dans la définition d'une fonction récursive, il faut toujours prévoir au moins un ***cas d'arrêt***, dans lequel la fonction ne s'appelle pas elle-même, sinon elle ne fera que s'appeler elle-même indéfiniment. Dans l'exemple précédent, c'est le cas **If n=0**.
- Il est toujours possible d'écrire aussi une version itérative de la fonction :



II 3 Définir une fonction récursive

Règles à suivre

1. Écrire l'en-tête de la fonction.
2. Vérifier tout d'abord que la fonction est adaptée à une définition récursive, autrement dit que l'on sait calculer une valeur à partir d'un appel plus simple à la même fonction.
3. Prévoir la condition d'arrêt.
4. Dans tous les appels récursifs, s'assurer que les arguments sont plus simples que ceux avec lesquels la fonction a été appelée : nombres plus petits, chaînes de caractères plus courtes, etc.
5. Déterminer la valeur de retour de la fonction.

Exercice :
Ecrire une fonction récursive en TI-Basic qui calcule le **quotient** de la division euclidienne d'un nombre entier par un autre.

1. Le *dividende* et le *diviseur* sont les deux arguments, et la valeur de retour est le *quotient*. On a donc l'entête :

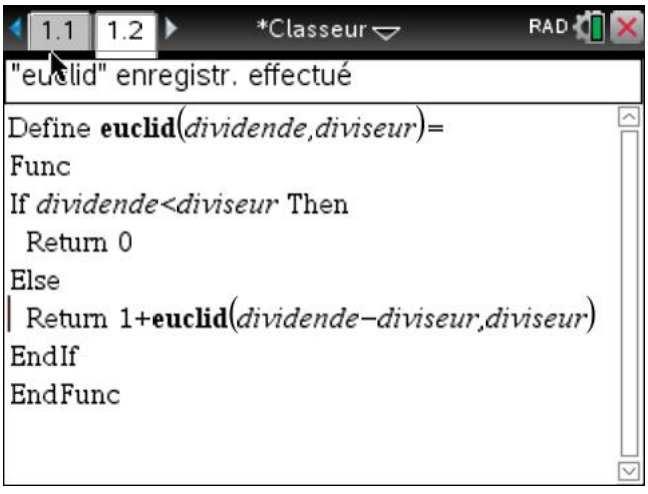
```
Define euclid(dividende,diviseur)=
```

2. Une définition récursive est possible : on diminue le *dividende* à chaque appel récursif jusqu'à avoir compté combien de fois il contient le *diviseur*.
3. La condition d'arrêt est le cas où le *dividende* est inférieur au *diviseur* : le *quotient* est alors nul. La fonction commence donc par le test :

```
If dividende<diviseur Then
  Return 0
```

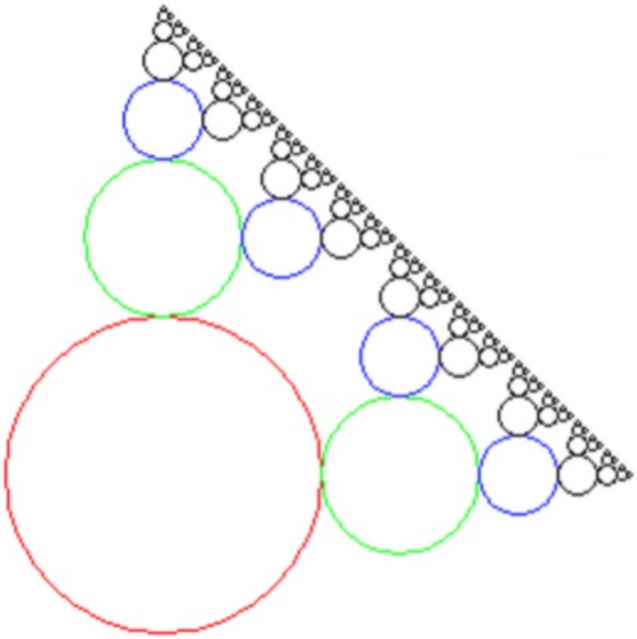
4. Dans les autres cas, le *dividende* est supérieur au *diviseur*, on retranche le *diviseur* au *dividende*, le *quotient* dont donc augmenter de 1. Le *diviseur* n'est pas modifié.

La fonction s'écrit donc :



II 4 Exemple : des images récursives

La récursivité est un outil utile en géométrie algorithmique et en synthèse d'images. Observons par exemple le dessin suivant :



Une manière de le décrire est de dire qu'il est formé :
- d'un cercle, en rouge sur la figure ;
- puis de deux cercles tangents de rayon moitié, en vert, sur la figure ;
- puis pour chacun des cercles verts, de deux cercles tangents de rayon moitié, en bleu sur la figure
- etc.

Mais une autre manière de le décrire est de dire qu'il est formé d'un cercle, en rouge sur la figure, et de deux copies plus petites du dessin lui-même (en vert) et ainsi de suite (en bleu puis en noir)...
Cela nous montre comment écrire une fonction récursive qui fait ce dessin.

```
1  -- la fenetre est environ de 300 par 200
2
3  -- La fonction de rafraichissement d'écran
4  function on.paint(gc)
5      gc:setColorRGB(0,0,255)
6      cercle(gc, 100 , 100, 40)
7  end
8
9  --fonction de construction recursive
10 function cercle(gc, x , y , r)
11
12     -- un cercle
13     gc:drawArc(x, y,2*r,2*r,0,360)
14
15     if r > 1 then
16         -- deux cercles de rayon moitié
17         cercle(gc,x+2*r,y+r/2,r/2)
18         cercle(gc,x+r/2,y - r ,r/2)
19     end
20 end
```