TP4 : boucles REPEAT et WHILE

ECE3 Lycée Carnot

5 novembre 2010

Nous avons vu lors du TD précédent, en étudiant les boucles FOR, une première façon de faire des instructions répétitives en Pascal. C'est un bon début, mais les boucles FOR présentent un défaut : il faut savoir à l'avance le nombre d'étapes du calcul, ce qui ne pose aucun problème quand il s'agit par exemple de calculer le terme d'indice n d'une suite récurrente, mais est beaucoup plus gênant quand on a envie de faire certains calculs jusqu'à ce qu'un évènement se produise. Exemple classique, on veut faire jouer l'utilisateur au jeu suivant : l'ordinateur tire un nombre au hasard entre 0 et 100, et l'utilisateur doit le deviner en utilsant le moins d'essais possible, sachant qu'à chaque essai l'ordinateur lui précise si le nombre tenté est trop grand ou trop petit. On souhaite donc que l'utilisateur propose des nombres jusqu'à ce qu'il soit tombé sur le bon nombre, on ne sait pas à l'avance le nombre d'essais nécessaire.
Les boucles REPEAT et WHILE servent donc, tout comme les boucles FOR, à faire des instructions répétitives, mais l'arrêt de la boucle sera déterminé par une condition et non plus par un nombre d'étapes donné. La différence entre les deux instructions est minime : avec une boucle REPEAT, l'instruction est effectuée avant que la condition ne soit testée, alors qu'avec une boucle WHILE, le test a lieu d'abord. Leur syntaxe respective est la suivante :

REPEAT instruction UNTIL condition ;
WHILE condition DO instruction ;

Il est souvent utile malgré tout de compter le nombre d'étapes dans une boucle de type REPEAT ou WHILE. Pour cela, la méthode consiste à créer une variable entière qui servira de compteur, à l'initialiser à la valeur 0 et à augmenter sa valeur d'une unité à chaque passage de la boucle. Par exemple, le programme suivant calcule la valeur du plus petit entier pour lequel $n! > 1000$:

```
PROGRAM tagada ;
VAR p,i : integer ;
BEGIN
i :=0 ; p :=1 ;
WHILE p<1000 DO
BEGIN
i :=i+1 ;
p :=p*i ;
END ;
WriteLn(i) ;
END.
```

Si on remplaçait la boucle WHILE par une boucle REPEAT, on calculerait exactement la même chose, mais on simplifieerait légèrement la syntaxe du programme, car on n'aurait pas besoin du BEGIN END ; qui a été nécessaire avec la boucle WHILE : en effet, tout ce qui se trouve entre les mots-clés REPEAT et UNTIL est effectué à chaque tour de boucle, quel que soit le nombre d'instructions.

Petits exercices

1. Écrire un programme permettant de jouer au jeu décrit un peu plus haut. Pour tirer un nombre aléatoire, on dispose de l'instruction **random(100)**, qui permet exactement de tirer un entier aléatoire entre 0 et 99. Par contre, pour que cette instruction fonctionne, il faut insérer la commande **Randomize ;** auparavant dans le programme (par exemple juste après le begin). Compléter le programme pour qu'il compte le nombre d'essais effectués avant de trouver la bonne réponse. Question subsidaire : quelle est la meilleure tactique à ce jeu ? Combien d'essais aupire mettra-t-on à trouver le nombre avec cette tactique ?
2. On note $S_n = \sum_{k=1}^{k=n} \frac{1}{k}$. On admet que $\lim_{n \to +\infty} S_n = +\infty$ (depuis le temps qu'on voit cette somme, vous allez finir par le savoir). Écrire un programme calculant la plus petite valeur de n pour laquelle $S_n > 5$, puis $S_n > 10$.
3. La suite de Syracuse est définie de la façon suivante : u_0 est un entier naturel différent de 0, et ensuite, on a $u_{n+1} = \frac{u_n}{2}$ si u_n est pair (j'ai bien dit si u_n est pair, et pas si n est pair), et $u_{n+1} = 3u_n + 1$ si u_n est impair. Vérifier à la main sur quelques exemples que la suite finit par prendre la valeur 1 (et est ensuite périodique). Écrire un programme calculant la plus petite valeur de n pour laquelle $u_n = 1$ (u_0 étant choisi par l'utilisateur). Modifier le programme pour qu'il calcule également la plus grande valeur prise par la suite.
4. On a vu en cours que les deux suites définies par $u_n = \sum_{k=1}^{k=n} \frac{1}{k^2}$ et $v_n = u_n + \frac{1}{n}$ étaient adjacentes. Écrire un programme calculant une valeur approchée de leur limite commune à ε près, ε étant choisi par l'utilisateur.