Partie 5
Les Boucles

« Les premiers 90% du code prennent les premiers 90% du temps de développement. Les 10% restants prennent les autres 90% du temps de développement »
Tom Cargill

Et ça y est, on y est, on est arrivés, la voilà, c'est Broadway, la quatrième et dernière structure : ça est les **boucles**. Si vous voulez épater vos amis, vous pouvez également parler de **structures répétitives**, voire carrément de **structures itératives**. Ca calme, hein ? Bon, vous faites ce que vous voulez, ici on est entre nous, on parlera de boucles.

Les boucles, c'est généralement le point douloureux de l'apprenti programmeur. C'est là que ça coince, car autant il est assez facile de comprendre comment fonctionnent les boucles, autant il est souvent long d'acquérir les réflexes qui permettent de les élaborer judicieusement pour traiter un problème donné.

On peut dire en fait que les boucles constituent la seule vraie structure logique caractéristique de la programmation. Si vous avez utilisé un tableur comme Excel, par exemple, vous avez sans doute pu manier des choses équivalentes aux variables (les cellules, les formules) et aux tests (la fonction SI…). Mais les boucles, ça, ça n'a aucun équivalent. Cela n'existe que dans les langages de programmation proprement dits.

Le maniement des boucles, s'il ne différencie certes pas l'homme de la bête (il ne faut tout de même pas exagérer), est tout de même ce qui sépare en informatique le programmeur de l'utilisateur, même averti.

Alors, à vos futures – et inévitables - difficultés sur le sujet, il y a trois remèdes : de la rigueur, de la patience, et encore de la rigueur !

1. A quoi cela sert-il donc ?

Prenons le cas d'une saisie au clavier (une lecture), où par exemple, le programme pose une question à laquelle l'utilisateur doit répondre par O (Oui) ou N (Non). Mais tôt ou tard, l'utilisateur, facétieux ou maladroit, risque de taper autre chose que la réponse attendue. Dès lors, le programme peut planter soit par une erreur d'exécution (parce que le type de réponse ne correspond pas au type de la variable attendu) soit par une erreur fonctionnelle (il se déroule normalement jusqu'au bout, mais en produisant des résultats fantaisistes).

Alors, dans tout programme un tant soit peu sérieux, on met en place ce qu'on appelle un **contrôle de saisie**, afin de vérifier que les données entrées au clavier correspondent bien à celles attendues par l'algorithme.

A vue de nez, on pourrait essayer avec un SI. Voyons voir ce que ça donne :

```
Variable Rep en Caractère
Début
      Ecrire "Voulez-vous un café ? (O/N)"
      Lire Rep
      Si Rep <> "O" ET Rep <> "N" Alors
```

```
          Ecrire "Saisie erronée. Recommencez"
          Lire Rep
      FinSi
Fin
```

C'est impeccable. Du moins tant que l'utilisateur a le bon goût de ne se tromper qu'une seule fois, et d'entrer une valeur correcte à la deuxième demande. Si l'on veut également bétonner en cas de deuxième erreur, il faudrait rajouter un SI. Et ainsi de suite, on peut rajouter des centaines de SI, et écrire un algorithme aussi lourd qu'une blague des Grosses Têtes, on n'en sortira pas, il y aura toujours moyen qu'un acharné flanque le programme par terre.

La solution consistant à aligner des SI… en pagaille est donc une impasse. La seule issue est donc de flanquer une **structure de boucle**, qui se présente ainsi :

```
TantQue booléen
     …
     Instructions
     …
FinTantQue
```

Le principe est simple : le programme arrive sur la ligne du TantQue. Il examine alors la valeur du booléen (qui, je le rappelle, peut être une variable booléenne ou, plus fréquemment, une condition). Si cette valeur est VRAI, le programme exécute les instructions qui suivent, jusqu'à ce qu'il rencontre la ligne FinTantQue. Il retourne ensuite sur la ligne du TantQue, procède au même examen, et ainsi de suite. Le manège enchanté ne s'arrête que lorsque le booléen prend la valeur FAUX.

Illustration avec notre problème de contrôle de saisie. Une première approximation de la solution consiste à écrire :

```
Variable Rep en Caractère
Début
      Ecrire "Voulez-vous un café ? (O/N)"
      TantQue Rep <> "O" ET Rep <> "N"
         Lire Rep
      FinTantQue
Fin
```

Là, on a le squelette de l'algorithme correct. Mais de même qu'un squelette ne suffit pas pour avoir un être vivant viable, il va nous falloir ajouter quelques muscles et organes sur cet algorithme pour qu'il fonctionne correctement.

Son principal défaut est de provoquer une erreur à chaque exécution. En effet, l'expression booléenne qui figure après le TantQue interroge la valeur de la variable Rep. Malheureusement, cette variable, si elle est été déclarée, n'a pas été affectée avant l'entrée dans la boucle. On teste donc une variable qui n'a pas de valeur, ce qui provoque l'arrêt immédiat de l'exécution. Pour éviter ceci, on n'a pas le choix : il faut que la variable Rep ait déjà été affectée avant qu'on en arrive au premier tour de boucle. Pour cela, on peut faire une première lecture de Rep avant la boucle. Dans ce cas, celle-ci ne servira qu'en cas de mauvaise saisie lors de cette première lecture. L'algorithme devient alors :

```
Variable Rep en Caractère
Début
      Ecrire "Voulez-vous un café ? (O/N)"
      Lire Rep
      TantQue Rep <> "O" ET Rep <> "N"
          Lire Rep
      FinTantQue
Fin
```

Une autre possibilité, fréquemment employée, consiste à ne pas lire, mais à affecter arbitrairement la variable avant la boucle. Arbitrairement ? Pas tout à fait, puisque cette affectation doit avoir pour résultat de provoquer l'entrée obligatoire dans la boucle. L'affectation doit donc faire en sorte que le booléen soit mis à VRAI pour déclencher le premier tour de la boucle. Dans notre exemple, on peut donc affecter Rep avec n'importe quelle valeur, hormis « O » et « N » : car dans ce cas, l'exécution sauterait la boucle, et Rep ne serait pas du tout lue au clavier. Cela donnera par exemple :

```
Variable Rep en Caractère
Début
      Rep ← "X"
      Ecrire "Voulez-vous un café ? (O/N)"
      TantQue Rep <> "O" ET Rep <> "N"
          Lire Rep
      FinTantQue
Fin
```

Cette manière de procéder est à connaître, car elle est employée très fréquemment.

Il faut remarquer que les deux solutions (lecture initiale de Rep en dehors de la boucle ou affectation de Rep) rendent toutes deux l'algorithme satisfaisant, mais présentent une différence assez importante dans leur structure logique.

En effet, si l'on choisit d'effectuer une lecture préalable de Rep, la boucle ultérieure sera exécutée uniquement dans l'hypothèse d'une mauvaise saisie initiale. Si l'utilisateur saisit une valeur correcte à la première demande de Rep, l'algorithme passera sur la boucle sans entrer dedans.

En revanche, avec la deuxième solution (celle d'une affectation préalable de Rep), l'entrée de la boucle est forcée, et l'exécution de celle-ci, au moins une fois, est rendue obligatoire à chaque exécution du programme. Du point de vue de l'utilisateur, cette différence est tout à fait mineure ; et à la limite, il ne la remarquera même pas. Mais du point de vue du programmeur, il importe de bien comprendre que les cheminements des instructions ne seront pas les mêmes dans un cas et dans l'autre.

Pour terminer, remarquons que nous pourrions peaufiner nos solutions en ajoutant des affichages de libellés qui font encore un peu défaut. Ainsi, si l'on est un programmeur zélé, la première solution (celle qui inclut deux lectures de Rep, une en dehors de la boucle, l'autre à l'intérieur) pourrait devenir :

```
Variable Rep en Caractère
Début
```

```
      Ecrire "Voulez-vous un café ? (O/N)"
      Lire Rep
      TantQue Rep <> "O" ET Rep <> "N"
          Ecrire "Vous devez répondre par O ou N. Recommencez"
          Lire Rep
      FinTantQue
      Ecrire "Saisie acceptée"
Fin
```

Quant à la deuxième solution, elle pourra devenir :

```
Variable Rep en Caractère
Début
      Rep ← "X"
      Ecrire "Voulez-vous un café ? (O/N)"
      TantQue Rep <> "O" ET Rep <> "N"
           Lire Rep
           Si Rep <> "O" ET Rep <> "N" Alors
                Ecrire "Saisie Erronée, Recommencez"
           FinSi
      FinTantQue
Fin
```

Bon, eh bien vous allez pouvoir faire de chouettes algorithmes, déjà rien qu'avec ça…

Réalisez les exercices 1 à 3

LE GAG DE LA JOURNEE

C'est d'écrire une structure TantQue dans laquelle le booléen n'est jamais VRAI. Le programme ne rentre alors jamais dans la superbe boucle sur laquelle vous avez tant sué !

Mais la faute symétrique est au moins aussi désopilante.

Elle consiste à écrire une boucle dans laquelle le booléen ne devient jamais FAUX. L'ordinateur tourne alors dans la boucle comme un dératé et n'en sort plus. Seule solution, quitter le programme avec un démonte-pneu ou un bâton de dynamite. La « **boucle infinie** » est une des hantises les plus redoutées des programmeurs. C'est un peu comme le verre baveur, le poil à gratter ou le bleu de méthylène : c'est éculé, mais ça fait toujours rire.

Cette faute de programmation grossière – mais fréquente - ne manquera pas d'égayer l'ambiance collective de cette formation… et accessoirement d'étancher la soif proverbiale de vos enseignants.

2. Boucler en comptant, ou compter en bouclant

Dans le dernier exercice, vous avez remarqué qu'une boucle pouvait être utilisée pour augmenter la valeur d'une variable. Cette utilisation des boucles est très fréquente, et dans ce cas, il arrive très souvent qu'on ait besoin d'effectuer un nombre déterminé de passages. Or, a priori, notre structure TantQue ne sait pas à l'avance combien de tours de boucle elle va effectuer (puisque le nombre de tours dépend de la valeur d'un booléen).

C'est pourquoi une autre structure de boucle est à notre disposition :

```
Variable Truc en Entier
Début
      Truc ⟵ 0
      TantQue Truc < 15
           Truc ⟵ Truc + 1
          Ecrire "Passage numéro : " & Truc
      FinTantQue
Fin
```

Equivaut à :

```
Variable Truc en Entier
Début
      Pour Truc ⟵ 1 à 15
          Ecrire "Passage numéro : " & Truc
      Truc Suivant
Fin
```

Insistons : **la structure « Pour … Suivant » n'est pas du tout indispensable** ; on pourrait fort bien programmer toutes les situations de boucle uniquement avec un « Tant Que ». Le seul intérêt du « Pour » est d'épargner un peu de fatigue au programmeur, en lui évitant de gérer lui-même la progression de la variable qui lui sert de compteur (on parle d'**incrémentation**, encore un mot qui fera forte impression).

Dit d'une autre manière, la structure « Pour … Suivant » est un cas particulier de TantQue : celui où le programmeur peut dénombrer à l'avance le nombre de tours de boucles nécessaires.

Il faut noter que dans une structure « Pour … Suivant », la progression du compteur est laissée à votre libre disposition. Dans la plupart des cas, on a besoin d'une variable qui augmente de 1 à chaque tour de boucle. On ne précise alors rien à l'instruction « Pour » ; celle-ci, par défaut, comprend qu'il va falloir procéder à cette incrémentation de 1 à chaque passage, en commençant par la première valeur et en terminant par la deuxième.

Mais si vous souhaitez une progression plus spéciale, de 2 en 2, ou de 3 en 3, ou en arrière, de –1 en –1, ou de –10 en –10, ce n'est pas un problème : il suffira de le préciser à votre instruction « Pour » en lui rajoutant le mot « Pas » et la valeur de ce pas (Le « pas » dont nous parlons, c'est le « pas » du marcheur, « step » en anglais).

Naturellement, quand on stipule un pas négatif dans une boucle, la valeur initiale du compteur doit être supérieure à sa valeur finale si l'on veut que la boucle tourne ! Dans le cas contraire, on aura simplement écrit une boucle dans laquelle le programme ne rentrera jamais.

Nous pouvons donc maintenant donner la formulation générale d'une structure « Pour ». Sa syntaxe générale est :

```
Pour Compteur ⟵ Initial à Final Pas ValeurDuPas
    …
    Instructions
```

```
    …
Compteur suivant
```

Les structures **TantQue** sont employées dans les situations où l'on doit procéder à un traitement systématique sur les éléments d'un ensemble dont on ne connaît pas d'avance la quantité, comme par exemple :

- Le contrôle d'une saisie
- La gestion des tours d'un jeu (tant que la partie n'est pas finie, on recommence)
- La lecture des enregistrements d'un fichier (cf. Partie 10)

Les structures **Pour** sont employées dans les situations où l'on doit procéder à un traitement systématique sur les éléments d'un ensemble dont on connaît d'avance la quantité.

Nous verrons dans les chapitres suivants des séries d'éléments appelés tableaux (parties 6 et 8) et chaînes de caractères (partie 9). Selon les cas, le balayage systématique des éléments de ces séries pourra être effectué par un Pour ou par un TantQue : tout dépend si la quantité d'éléments à balayer (donc le nombre de tours de boucles nécessaires) peut être dénombrée à l'avance par le programmeur ou non.

3. Des boucles dans des boucles

(« *Tout est dans tout, et réciproquement* »)

On rigole, on rigole !

De même que les poupées russes contiennent d'autres poupées russes, de même qu'une structure SI … ALORS peut contenir d'autres structures SI … ALORS, une boucle peut tout à fait contenir d'autres boucles. Y a pas de raison.

```
Variables Truc, Trac en Entier
Début
      Pour Truc ← 1 à 15
            Ecrire "Il est passé par ici"
            Pour Trac ← 1 à 6
                  Ecrire "Il repassera par là"
            Trac Suivant
      Truc Suivant
Fin
```

Dans cet exemple, le programme écrira une fois "il est passé par ici" puis six fois de suite "il repassera par là", et ceci quinze fois en tout. A la fin, il y aura donc eu 15 x 6 = 90 passages dans la deuxième boucle (celle du milieu), donc 90 écritures à l'écran du message « il repassera par là ». Notez la différence marquante avec cette structure :

```
Variables Truc, Trac en Entier
Début
      Pour Truc ← 1 à 15
            Ecrire "Il est passé par ici"
      Truc Suivant
```

```
      Pour Trac ← 1 à 6
            Ecrire "Il repassera par là"
      Trac Suivant
Fin
```

Ici, il y aura quinze écritures consécutives de "il est passé par ici", puis six écritures consécutives de "il repassera par là", et ce sera tout.

Des boucles peuvent donc être **imbriquées** (cas n°1) ou **successives** (cas n°2). Cependant, elles ne peuvent jamais, au grand jamais, être croisées. Cela n'aurait aucun sens logique, et de plus, bien peu de langages vous autoriseraient ne serait-ce qu'à écrire cette structure aberrante.

```
Variables Truc, Trac en Entier
Début
      Pour Truc ← …
            instructions
            Pour Trac ← …
                  instructions
            Truc Suivant
            instructions
      Trac Suivant
Fin
```

Pourquoi imbriquer des boucles ? Pour la même raison qu'on imbrique des tests. La traduction en bon français d'un test, c'est un « cas ». Eh bien un « cas » (par exemple, « est-ce un homme ou une femme ? ») peut très bien se subdiviser en d'autres cas (« a-t-il plus ou moins de 18 ans ? »).

De même, une boucle, c'est un traitement systématique, un examen d'une série d'éléments un par un (par exemple, « prenons tous les employés de l'entreprise un par un »). Eh bien, on peut imaginer que pour chaque élément ainsi considéré (pour chaque employé), on doive procéder à un examen systématique d'autre chose (« prenons chacune des commandes que cet employé a traitées »). Voilà un exemple typique de boucles imbriquées : on devra programmer une boucle principale (celle qui prend les employés un par un) et à l'intérieur, une boucle secondaire (celle qui prend les commandes de cet employé une par une).

Dans la pratique de la programmation, la maîtrise des boucles imbriquées est nécessaire, même si elle n'est pas suffisante. Tout le contraire d'Alain Delon, en quelque sorte.

4. Et encore une bêtise à ne pas faire !

Examinons l'algorithme suivant :

```
Variable Truc en Entier
Pour Truc ⟵ 1 à 15
     Truc ⟵ Truc * 2
     Ecrire "Passage numéro : " & Truc
Truc Suivant
```

Vous remarquerez que nous faisons ici gérer « en double » la variable Truc, ces deux gestions étant contradictoires. D'une part, la ligne « Pour… » augmente la valeur de Truc de 1 à chaque passage. D'autre part la ligne « Truc ⟵ Truc * 2 » double la valeur de Truc à chaque passage. Il va sans dire que de telles manipulations perturbent complètement le déroulement normal de la boucle, et sont causes, sinon de plantages, tout au moins d'exécutions erratiques.

LE GAG DE LA JOURNEE

Il consiste donc à manipuler, au sein d'une boucle **Pour**, la variable qui sert de compteur à cette boucle. Cette technique est à proscrire absolument… sauf bien sûr, si vous cherchez un prétexte pour régaler tout le monde au bistrot.

Mais dans ce cas, n'ayez aucune inhibition, proposez-le directement, pas besoin de prétexte.

Réalisez les exercices 4 à 10