ALGORITHMIQUE

Partie 5 – Les boucles

Rappel : il n'existe que 4 briques :

L'AFFECTATION DE VARIABLES

LA LECTURE / ECRITURE

LES TESTS

LES BOUCLES

On peut également parler de **structures répétitives**, ou de **structures itératives**.

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.

Une structure de boucle 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 peut être une variable booléenne ou 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 ne s'arrête que lorsque le booléen prend la valeur FAUX.

Illustration avec un 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

Cet algorithme provoque 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 a 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 une erreur et 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. 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. 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

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

Une erreur à éviter :

Elle consiste à écrire une boucle dans laquelle le booléen ne devient jamais FAUX. L'ordinateur tourne alors dans la boucle et n'en sort plus. Seule solution, quitter le programme à la sauvage.

Exercices

Exercice 5.1

Écrire un algorithme qui demande à l'utilisateur un nombre compris entre 1 et 3 jusqu'à ce que la réponse convienne.

Exercice 5.2

Écrire un algorithme qui demande un nombre compris entre 10 et 20, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, on fera apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

Exercice 5.3

Écrire un algorithme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Boucler en comptant

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 :

Structure en TANT QUE :

Variable Truc en Entier

Début

Truc $\leftarrow$ 0

TantQue Truc < 15

Truc $\leftarrow$ Truc + 1

Ecrire "Passage numéro : ", Truc

FinTantQue

Fin

Structure en POUR :

Variable Truc en Entier

Début

Pour Truc $\leftarrow$ 1 à 15

Ecrire "Passage numéro : ", Truc

Truc Suivant

Fin

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.

Mais si on souhaite une progression différente (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.

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

Pour Compteur $\leftarrow$ Initial à Final Pas ValeurDuPas

En conclusion :

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

Exercice :

Ecrire un algorithme utilisant une structure en POUR qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme affichera les nombres de 18 à 27.

Des boucles dans des boucles

De même qu'une structure SI ... ALORS peut contenir d'autres structures SI ... ALORS, une boucle peut tout à fait contenir d'autres boucles.

Variables Truc, Trac en Entier

Début

Pour Truc ← 1 à 10

 Ecrire "Il est passé par ici"

 Pour Trac ← 1 à 5

 Ecrire "Il repassera par là"

 Trac Suivant

Truc Suivant

Fin

Dans cet exemple, le programme écrira une fois "il est passé par ici" puis 5 fois de suite "il repassera par là", et ceci 10 fois en tout.

A la fin, il y aura donc eu $10 \times 5 = 50$ passages dans la deuxième boucle (celle du milieu), donc 50 écritures à l'écran du message « il repassera par là ».

Notez la différence avec cette structure :

Variables Truc, Trac en Entier

Début

Pour Truc ← 1 à 10

 Ecrire "Il est passé par ici"

Truc Suivant

Pour Trac ← 1 à 5

 Ecrire "Il repassera par là"

Trac Suivant

Fin

Ici, il y aura 10 écritures consécutives de "il est passé par ici", puis 5 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 être croisées.

Dans la pratique de la programmation, la maîtrise des boucles imbriquées est nécessaire.