La boucle Pour

1 La boucle Pour

On peut calculer la somme des entiers de 1 à 100 à l'aide d'une structure itérative `tant que` et de deux variables, `i` pour énumérer les entiers successifs de 1 à 100, et `S` pour les cumuler.

```
S := 0
i := 1
{S = ∑_{k=1}^{i-1} k}
tant que i≤100 faire
  S := S+i
  {S = ∑_{k=1}^{i} k}
  i := i+1
  {S = ∑_{k=1}^{i-1} k}
fin tant que
{S = ∑_{k=1}^{100} k}
```

Compte tenu de la condition, il est possible de dire avant l'exécution de ce `tant que` que la séquence d'instructions qu'il contient sera effectuée 100 fois.

De manière plus générale, si un algorithme s'écrit de cette façon

```
i := a
tant que i≤b faire
  (* séquence d'instructions *)
  i := i+1
fin tant que
```

et que la séquence d'instructions précédant l'incrémentation de la variable `i` ne modifie pas la valeur de cette variable, alors le nombre de fois que la séquence d'instructions sera exécutée est égal à `b-a+1`.

De telles itérations dont le nombre d'étapes est déterminée par une variable qui parcourt l'ensemble des valeurs comprises entre deux bornes est appelée une boucle `Pour`, et la variable est appelée *indice* de la boucle.

On peut réécrire le calcul de la somme des entiers compris entre 1 et 100 en utilisant une boucle **pour**.

```
S := 0
pour i variant de 1 à 100 faire
  {S = ∑_{k=1}^{i-1} k}
  S := S+i
  {S = ∑_{k=1}^{i} k}
fin pour
{S = ∑_{k=1}^{100} k}
```

Remarque : L'algorithme de calcul de la somme des entiers compris entre 1 et 100 donné ci–dessus procède en énumérant ces entiers en commençant par 1 et en terminant à 100. On peut bien entendu commencer par 100 et terminer par 1.

```
S := 0
i := 100
{S = ∑_{k=i+1}^{100} k}
tant que i>=1 faire
  S := S+i
  {S = ∑_{k=i}^{100} k}
  i := i-1
```

$\{S = \sum_{k=i+1}^{100} k\}$
fin tant que
$\{S = \sum_{k=1}^{100} k\}$

Les mêmes remarques que celles formulées plus haut s'appliquent à cette boucle `tant que` s'appliquent, et il est possible de la remplacer par une boucle **pour** mais avec un indie de boucle qui décroît.

```
S := 0
pour i variant de 100 à 1 décroissant faire
  {S = ∑_{k=i+1}^{100} k}
  S := S+i
  {S = ∑_{k=i}^{100} k}
fin pour
{S = ∑_{k=1}^{100} k}
```

2 La boucle Pour en Pascal

2.1 Boucles Pour à indice croissant

Toute séquence de deux instructions de la forme

```
i := a;
while i <= b do
begin
  (* séquence d'instructions *)
  i := i+1;
end {while};
```

peut être remplacée par une boucle **for** à indice croissant

```
for i := a to b do
begin
  (* séquence d'instructions *)
end {for};
```

Exemple : Calcul de la somme des entiers de 1 à 100

```
S := 0;
for i := 1 to 100 do
begin
  S := S+i;
end {for};
```

2.2 Boucles Pour à indice décroissant

Toute séquence de deux instructions de la forme

```
i := a;
while i >= b do
begin
  (* séquence d'instructions *)
  i := i-1;
end {while};
```

peut être remplacée par une boucle **for** décroissant

```
for i := a downto b do
begin
  (* séquence d'instructions *)
end {for};
```

Exemple : Calcul de la somme des entiers de 1 à 100

```
S := 0;
for i := 100 downto 1 do
begin
  S := S+i;
end {for};
```

2.3 Remarques sur l'indice de boucle

L'indice de boucle d'une boucle **for** est une variable, et à ce titre doit être déclarée[1].
Le compilateur Free Pascal ne permet pas la modification explicite de l'indice d'une boucle.

```
// EXEMPLE A NE PAS SUIVRE
for i := 1 to 10 do
begin
  i := i+1;
end {for};
```

avec le message d'erreur `Error : Illegal assignment to for-loop variable "i"`

Cependant, le compilateur n'est pas capable de détecter les modifications susceptibles d'être provoquées par effet de bord d'un appel à une procédure. Par exemple, si `p` est la procédure qui modifie une variable globale `i`

```
procedure p;
begin
  i := i+1;
end {p};
```

alors le compilateur accepte le code

```
// EXEMPLE A NE PAS SUIVRE
for i := 1 to 10 do
begin
  p;
  writeln(i);
end {for};
```

et à l'exécution, on obtient

```
2
4
6
8
10
```

On constate donc que l'indice de la boucle n'a pris que cinq valeurs au lieu de 10.

C'est une mauvaise idée que d'utiliser la valeur d'un indice de boucle en dehors de cette boucle.

Conclusion : Lors de l'écriture d'une boucle **pour**, utiliser comme indice de boucle une variable qui ne joue aucun rôle ailleurs et définie le plus localement possible.

3 Suites récurrentes

Soit $(u_n)_{n\in\mathbb{N}}$ une suite de nombres réels définie par son premier terme $u_0 = a$ ($a \in \mathbb{R}$) et pour tout $n \geq 0$ une relation de récurrence de la forme

$$u_{n+1} = f(u_n)$$

où f est une fonction réelle d'une variable réelle.

[1] ce n'est pas le cas dans tous les langages, cf le cas de Ada

Par exemple, on peut considérer la suite de réels définie par

$$\begin{aligned} u_0 &= 1 \\ u_{n+1} &= \frac{u_n}{2} + \frac{1}{u_n} \end{aligned}$$

on a ici $a = 1$ et $f(x) = \frac{x}{2} + \frac{1}{x}$

3.1 Calcul du terme d'indice n

On veut calculer le terme u_n lorsqu'est donné l'indice n.

```
u := a
{u = u_0}
pour i variant de 1 à n faire
  {u = u_{i-1}}
  u := f(u)
  {u = u_i}
fin pour
{u = u_n}
```

3.2 Calcul des termes d'indice 0 à n

Il faut distinguer le calcul de tous les termes avec ou sans mémorisation de ces termes.

Nous n'envisagerons ici que le calcul de tous les termes sans mémorisation, chaque terme calculé étant affiché. Pour le calcul avec mémorisation, il faudra disposer de la notion de tableaux qui sera abordée plus tard.

On veut donc une procédure qui affiche la valeur de tous les termes de la suite (u_n) d'indice 0 à n, n étant passé en paramètre.

Il suffit de calculer successivement chaque terme de la suite et afficher immédiatement sa valeur.

```
// affiche tous les termes u_k
// pour k compris entre 0 et n
procédure afficherTousTermes(n)
  u := a
  {u = u_0}
  écrire u
  pour i variant de 1 à n faire
    {u = u_{i-1}}
    u =f(u)
    {u = u_i}
    écrire u
  fin pour
fin procédure
```

3.3 Calcul du premier terme satisfaisant une condition

On peut aussi vouloir chercher le premier terme de la suite qui satisfait une condition donnée. Cette condition peut porter sur le dernier terme calculé, ou bien sur plusieurs termes déjà calculés.

```
u := a
{u = u_0}
tantque u ne satisfait pas la condition faire
  {u = u_{n-1} et ∀i < n  u_i ne satisfait pas la condition}
  u := f(u)
  {u = u_n et ∀i < n  u_i ne satisfait pas la condition}
fin tantque
{u = u_n est le premier terme de la suite qui satisfait la condition}
```

Remarque : si aucun terme de la suite ne satisfait la condition donnée, la boucle est infinie (le programme ne s'arrête pas). Aussi avant de concevoir de tels programmes est-il important de s'assurer qu'au moins un terme de la suite vérifie la condition.

3.4 Autres suites récurrentes

3.4.1 Suites récurrentes d'ordre plus élevé

ordre 2 : la suite de Fibonacci définie par ses deux premiers termes et une relation de récurrence d'ordre 2

$$\begin{aligned} u_0 &= 0 \\ u_1 &= 1 \\ u_{n+2} &= u_{n+1} + u_n \quad \forall n \in \mathbb{N} \end{aligned}$$

Facile à programmer

ordre total : la suite des nombres de Catalan définie par son premier terme et une relation de récurrence s'appuyant sur tous les termes qui précèdent

$$\begin{aligned} u_0 &= 1 \\ u_{n+1} &= \sum_{k=0}^{n} u_k u_{n-k} \quad \forall n \in \mathbb{N} \end{aligned}$$

difficile à programmer sans tableaux

4 Exercices

Exercice 1. *Tables de multiplication*

Réalisez un programme qui affiche une table de multiplication par un nombre donné par l'utilisateur. Une trace d'exécution du programme demandé est donnée ci-dessous :

```
Table de multiplication souhaitée : 7
 1 x 7 =  7
 2 x 7 = 14
 3 x 7 = 21
 ...
10 x 7 = 70
```

Exercice 2. *Calcul de puissances*

Réalisez un programme qui affiche la valeur de l'entier a^n pour deux entiers a et n donnés par l'utilisateur.

Une trace d'exécution du programme demandé est donnée ci-dessous :

```
a = 5
n = 3
5^3 = 125
```

Exercice 3. *Calcul de factorielles*

Réalisez un programme qui affiche la valeur de l'entier $n!$, n étant donné par l'utilisateur.

Une trace d'exécution du programme demandé est donnée ci-dessous :

```
n = 4
4! = 24
```

Exercice 4. *Somme de nombres lus au clavier*

Question 1.

Réalisez un programme qui lit une série de n nombres, l'entier n étant défini par l'utilisateur, et affiche ensuite la somme de ces nombres.

Une trace d'exécution du programme demandé est donnée ci-dessous :

```
n = 4
nombre 1 : 3
nombre 2 : 5
nombre 3 : 7
nombre 4 : 2
Total = 17
```

Question 2. Même chose mais pour déterminer le plus grand nombre entré.

Exercice 5. *Boucles imbriquées pour stars académiques*

Question 1.

Réalisez un programme qui affiche à l'écran une ligne de n étoiles, l'entier n étant donné par l'utilisateur.

Une trace d'exécution du programme demandé est donnée ci-dessous :

```
n = 5
*****
```

Question 2. Réalisez un programme qui affiche à l'écran un carré de $n \times n$ étoiles, l'entier n étant donné par l'utilisateur.

Une trace d'exécution du programme demandé est donnée ci-dessous :

```
n = 5
*****
*****
*****
*****
*****
```

Question 3. Réalisez un programme qui affiche à l'écran un triangle (isocèle) de hauteur n, l'entier n étant donné par l'utilisateur.

Une trace d'exécution du programme demandé est donnée ci-dessous :

```
n = 5
*
**
***
****
*****
```

Question 4. Réalisez un programme qui affiche à l'écran un triangle (isocèle) de hauteur n, l'entier n étant donné par l'utilisateur, pointe en bas.

Une trace d'exécution du programme demandé est donnée ci-dessous :

```
n = 5
*****
****
***
**
*
```

Exercice 6. **downto** *n'est pas indispensable*

Question 1. À l'aide d'une boucle à indices croissant, réécrivez l'instruction qui suit de sorte que l'affichage soit identique.

```
for i := 100 downto 1 do begin
  writeln(i);
end {for};
```

Question 2. En supposant que l'instruction `action(i)` accomplisse une certaine tâche qui dépend de i, comment récrire l'instruction

```
for i := b downto a do begin
  action(i);
end {for};
```

avec une boucle à indices croissant ?

Exercice 7. *Calcul de la suite de Heron*

Programmez le calcul des premiers termes de la suite de Heron définie par

$$\begin{aligned} u_0 &= a \\ u_{n+1} &= \frac{u_n}{2} + \frac{B}{2u_n} \end{aligned}$$

où a et B sont deux réels positifs.

Utilisez votre programme pour différentes valeurs de a et B.

Exercice 8. *Suite de Fibonacci*

Question 1. Programmez le calcul des premiers termes de la suite de Fibonacci. Faîtes le en utilisant le type `Cardinal`, puis le type `Real`. Calculez les termes jusqu'à l'indice 50. Comparez les résultats selon le type utilisé.

Question 2. Cette suite est croissante et tend vers $+\infty$ lorsque n tend vers $+\infty$. Écrivez une fonction qui calcule l'indice du premier terme supérieur ou égal à un réel positif s passé en paramètre.