

TP Informatique 1

Corrigé de la Série de TP N°5 – Les instructions itératives Pour, Tant-que et Répéter

Rappel :

Structures de contrôle répétitives : nous permettent de répéter un traitement un nombre fini de fois.

Nous avons trois types de structures itératives (boucles) :

1. Boucle Pour (For)

La structure de contrôle répétitive *Pour (For* en langage *PASCAL*) utilise un indice entier qui varie (avec un *incrément = I*) d'une valeur initiale jusqu'à une valeur finale. À la fin de chaque itération, l'indice est incrémenté de 1 d'une manière automatique (implicite).

La syntaxe de la *boucle pour* est comme suit :

```
pour <indice>←<vi> à <vf> faire           for <indice>:=<vi> to <vf> do  
|                                     <instruction(s)>      begin  
finPour;                                     <instruction(s)>;  
                                                end;
```

<indice> : variable entière
<vi> : valeur initiale <vf> : valeur finale

2. Boucle Tant-que (While)

La structure de contrôle répétitive *Tant-que (While* en langage *PASCAL*) utilise une expression *logique* ou *booléenne* comme condition d'accès à la boucle : *si* la condition est vérifiée (elle donne un résultat *vrai* : *TRUE*) donc on entre à la boucle, *sinon* on la quitte.

La syntaxe de la *boucle tant-que* est comme suit :

```
tant-que <condition> faire                 while <condition> do  
|                                     <instruction(s)>      begin  
finTant-que;                                     <instruction(s)>;  
                                                end;
```

<condition> : expression logique qui peut être vraie ou fausse.

On exécute le bloc d'instructions *tant-que* la condition est *vraie*. Une fois la condition est *fausse*, on arrête la boucle, et on continue l'exécution de l'instruction qui vient après *fin Tant-que* (après end).

3. Boucle Répéter (Repeat)

La structure de contrôle répétitive *Répéter (Repeat* en langage *PASCAL*) utilise une expression *logique* ou *booléenne* comme condition de sortie de la boucle : *si* la condition est vérifiée (elle donne un résultat *vrai* : *TRUE*) on sort de la boucle, *sinon* on y accède (on répète l'exécution du bloc).

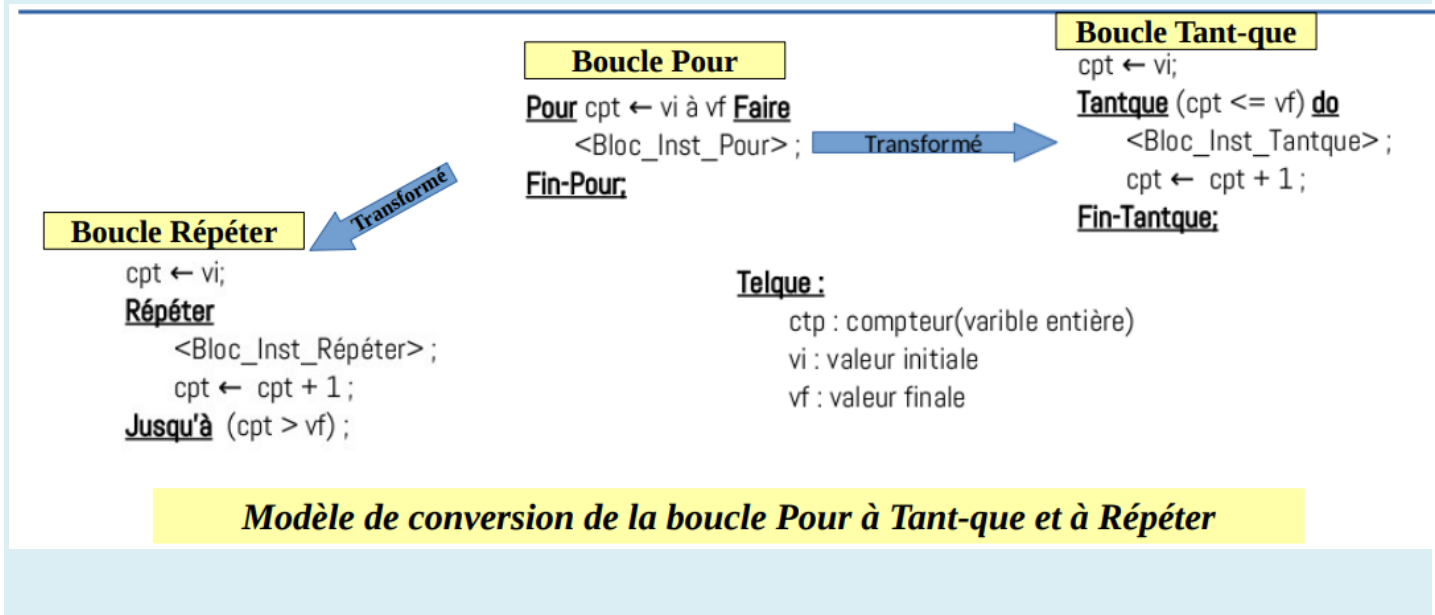
La syntaxe de la boucle répéter est comme suit :

```
répéter                                     repeat  
|                                     <instruction(s)>      until <condition>;  
jusqu'à <condition>;
```

<condition> : expression logique qui peut être vraie ou fausse.

On exécute le bloc d'instructions *jusqu'à* avoir la condition *correcte*. Une fois la condition est *vérifiée*, on *arrête la boucle*, et on continue l'exécution de l'instruction qui vient après *jusqu'à* (*après until*).

Dans la *boucle repeat* on n'utilise pas *begin* et *end* pour délimiter le bloc d'instructions (le bloc est déjà délimité par *repeat et until*).



Solution de l'exercice N°01 :

1- Traduction de l'algorithme en programme PASCAL :

Algorithme	Programme PASCAL
Algorithme Exo1 ; Variables X, P, S : réel ; i, N : entier ; Début {-*-*- Entrées -*-*-} Ecrire ('Donner les valeurs de N et X : '); Lire (N, X) ; {-*-*- Traitement -*-*-} S ← 0 ; P ← X ; Pour i ← 1 à N faire S ← S+P/i ; P ← P*X ; Fin-Pour {-*-*- Sorties -*-*-} Ecrire ('S = ', S:4:3) ; Fin.	Program exo1 ; Var X, P, S : real ; i, N : integer ; Begin {-*-*- Entrées -*-*-} Write ('Donner les valeurs de N et X : '); Read (N,X) ; {-*-*- Traitement -*-*-} S:=0; P:=X; For i:=1 to N do Begin S:=S+P/i; P:=P*X; End; {-*-*- Sorties -*-*-} Write ('S = ', S:4:3); End.

☞ *Compilation et exécution du programme*

```

1 Program exol ;
2 Var
3   X, P, S : real;
4   I, N : integer;
5 Begin
6   {*** Entrées ***}
7   Write('Donner les valeurs de N et X : ');
8   Read(N,X);
9   {*** Traitements ***}
10  S:=0;
11  P:=X;
12  For I:=1 to N do
13  Begin
14    S:=S+P/I;
15    P:=P*X;
16  End;
17  {*** Sorties ***}
18  Write('S =', S:4:3);
19 End.
  
```

Sélection MyPascal V1.20.5 (Exécution) C:...

Donner les valeurs de N et X : 3 3
S =16.500

Après l'exécution

2- Déroulement de l'algorithme pour N=3 et X =3 :

Instructions	Variables					Affichage
	N	X	i	P	S	
Lire (N,X);	3	3	/	/	/	/
S ← 0;	3	3	/	/	0	/
P ← X; P ← 3	3	3	/	3	0	/
Pour i=1 alors S ← S+ P/i ; S ← 0+ 3/1 ; S ← 3 ; P ← P*X ; P ← 3*3 ; P ← 9 ; Fin-Pour	3	3	1		3	
Pour i=2 alors S ← S+ P/i ; S ← 0+ 3/1+ 9/2 ; S ← 7.5 ; P ← P*X ; P ← 3*3*3 ; P ← 27 ; Fin-Pour	3	3	1		7.5	
Pour i=3 alors S ← S+ P/i ; S ← 0+ 3/1+ 9/2 + 27/3 ; S ← 16.5 ; P ← P*X ; P ← 3*3*3*3 ; P ← 81 ; Fin-Pour	3	3	1		16.5	
Ecrire ('S =', S:4:3) ;	3	3	1	81	16.5	S = 16.500

3- Dédution de l'expression finale :

Selon le déroulement ci-dessus, nous avons :

Pour $i = 1$, nous avons $S = 3$

Pour $i = 2$, nous avons $S = 3 + \frac{9}{2} = 7.5$

Pour $i = 3$, nous avons $S = 3 + \frac{9}{2} + \frac{27}{3} = 16.5$

Pour $i = N$ nous aurons : $S = X + \frac{X^2}{2} + \frac{X^3}{3} + \dots + \frac{X^N}{N}$

On peut généraliser par la formule suivante :

$$S = \sum_{i=1}^N \frac{X^i}{i} \quad \text{ou} \quad S = X + \frac{X^2}{2} + \frac{X^3}{3} + \dots + \frac{X^N}{N}$$

4) Réécriture de l'algorithme/PASCAL en remplaçant la boucle Pour par la boucle Tant-que.

Algorithme	Programme PASCAL
Algorithme Exo1 ; Variabes X, P, S : réel ; i, N : entier ; Début {-*-*- Entrées -*-*-} Ecrire ('Donner les valeurs de N et X : '); Lire (N, X); S ← 0 ; P ← X ; i ← 1 ; Tant-que i ≤ N faire S ← S+ P/i ; P ← P*X ; i ← i+1 ; Fin Tant-que {-*-*- Sortie -*-*-} Ecrire ('S =', S:4:3) ; Fin.	Program Exo1 ; Var X, P, S : real ; i, N : integer ; Begin {-*-*- Entrées -*-*-} Write ('Donner les valeurs de N et X : '); Read (N, X) ; S := 0 ; P := X ; i :=1 ; While i <=N do Begin S := S+ P/i ; P := P*X ; i := i+1 ; End; {-*-*- Sortie -*-*-} Write ('S =', S:4:3) ; End.

5) Réécriture de l'algorithme/PASCAL en remplaçant la boucle Pour par la boucle Répéter.

Algorithme	Programme PASCAL
Algorithme Exo1 ; Variables X, P, S : réel ; i, N : entier ; Début {-*-*- Entrées -*-*-} Ecrire ('Donner les valeurs de N et X : '); Lire (N, X) ; {-*-*- Traitement -*-*-} S ← 0 ; P ← X ; i ← 1 ; Répéter S ← S+ P/i ; P ← P*X ; i ← i+1 ; Jusqu'a i>N; {-*-*- Sortie -*-*-} Ecrire ('S =', S:4:3) ; Fin.	Program Exo1 ; Var X, P, S : real ; i, N : integer ; Begin {-*-*- Entrées -*-*-} Write ('Donner les valeurs de N et X : '); Read (N, X) ; {-*-*- Traitement -*-*-} S := 0 ; P := X ; i := 1 ; Repeat S := S+ P/i ; P := P*X ; i := i+1 ; Until i>N; {-*-*- Sortie -*-*-} Write ('S =', S:4:3) ; End.

6- Modification de l'algorithme pour Calculer la somme de f

Nous avons : $f = x + x^3/2 + x^5/3 + \dots + N^{\text{ième}} \text{ Terme.}$

$$f = x/1 + x^3/2 + x^5/3 + \dots$$

La somme peut s'écrire sous la forme généralisée suivante :

$$f = \sum_{i=1}^N \frac{X^{2i-1}}{i}$$

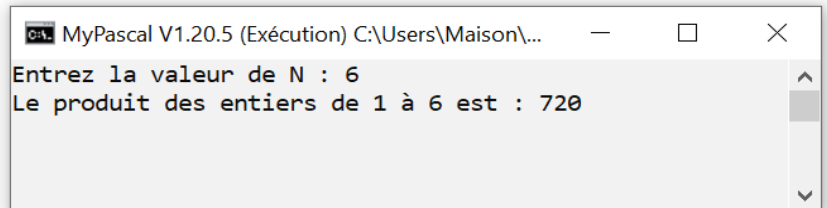
Algorithme TP5_Exo1; Variables i, N: entier ; X, f, P : réel; Début {-*-*- Entrées -*-*-} Écrire('Donner les valeurs de N et X : '); Lire(N, X) ; {-*-*- Traitements -*-*-} f ← 0 ; P ← 1/X; i ← 1; Pour i ← 1 à N faire P ← P * sqr(X); f ← f + P / i; Fin-Pour ; {-*-*- Sorties -*-*-} Écrire ('Le résultat de f=', f:0:3) ; Fin.
--

Solution de l'exercice N°02 :

☞ Programme en Pascal pour calculer le produit P_1 des entiers de 1 à N :

Compilation et exécution du programme pour N=6 :

```
program CalculProduit_P_1;
var
  N, i: integer;
  produit: longint; { Utilisation de longint pour gérer les grands résultats }
begin
  { Entrée de la valeur de N }
  Write('Entrez la valeur de N : ');
  Readln(N);
  { Initialisation du produit à 1, car 1 est le neutre pour la multiplication }
  produit := 1;
  { Calcul du produit des entiers de 1 à N }
  for i := 1 to N do
  begin
    produit := produit * i;
  end;
  { Affichage du résultat }
  Writeln('Le produit des entiers de 1 à ', N, ' est : ', produit);
end.
```



☞ Programme en Pascal qui calcule un produit P_2 de la formule :

$$P_2 = 1 \times (1+2) \times (1+2+3) \times \dots \times (1+2+3+\dots+N)$$

Autrement dit, chaque facteur dans le produit est une somme des premiers entiers jusqu'à un certain nombre, et vous multipliez tous ces résultats ensemble.

Décomposition du problème :

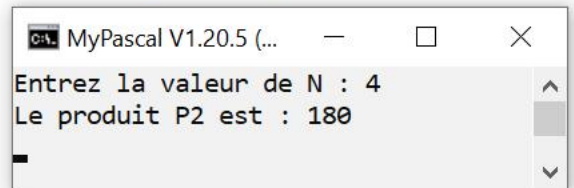
- Le premier terme est simplement 1.
- Le deuxième terme est 1+2.
- Le troisième terme est 1+2+3.
- Le N^{ième} terme est 1+2+...+k.

P_2 permet de multiplier tous ces termes jusqu'à N. La somme des premiers k entiers est donnée par la formule classique :

$$P_2(k) = \frac{k(k+1)}{2}, k=1, \dots, N;$$

Compilation et exécution du programme pour N=4 :

```
program CalculProduit_P_2;
var
  N, i, somme: integer;
  produit: longint; { Utilisation de longint pour éviter les débordements }
begin
  { Entrée de la valeur de N }
  Write('Entrez la valeur de N : ');
  Readln(N);
  { Initialisation du produit à 1, car 1 est l'élément neutre pour la multiplication }
  produit := 1; somme:=0;
  { Boucle pour calculer le produit des sommes des entiers }
  for i := 1 to N do
  begin
    { Calcul de la somme des entiers de 1 à i }
    {somme := (i * (i + 1)) div 2;} { Formule de la somme des premiers i entiers }
    somme := somme +i;
    ..... { Multiplier cette somme au produit total }
    produit := produit * somme;
  end;
  { Affichage du résultat }
  WriteLn('Le produit P2 est : ', produit);
end.
```



☞ Programme pour calculer la somme des factorielles des entiers de 1 à N, c'est-à-dire la somme :

$$S_1=1 +2! +3! + 4!+\dots+ N!$$

Décomposition de la fonction S_1 :

- Un nombre factoriel N! est défini comme le produit de tous les entiers de 1 à N.

Par exemple :

- $1!=1$
- $2!=2\times 1=2$
- $3!=3\times 2\times 1=6$
- $4!=4\times 3\times 2\times 1=24$

Le résultat de $S_1= 1+2+6+24= 33$

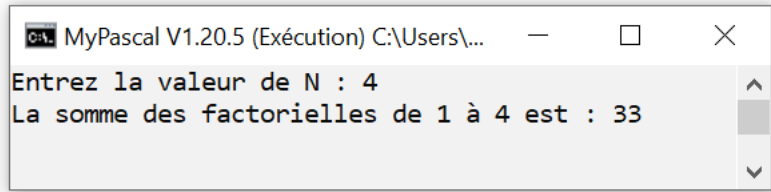
On utilise *longint* pour la variable *fact* et *somme*, car les factorielles de grands nombres peuvent rapidement devenir très grandes et dépasser la capacité d'un *integer* standard (qui est limité à 32,767).

☞ *Compilation et exécution du programme pour N=4*

```

program SommeFactorielle_S_1;
var
  N, i, j: integer;
  fact, somme: longint;
begin
  { Entrée de la valeur de N }
  Write('Entrez la valeur de N : ');
  Readln(N);
  { Initialisation de la somme à 0 }
  somme := 0; fact:=1;
  { Calcul de la somme des factorielle de 1! à N! }
  for i := 1 to N do
  begin
    { Initialisation de la factorielle pour i! }
    fact := fact * i;
    { Ajouter la factorielle de i à la somme totale }
    somme := somme + fact;
  end;
  { Affichage du résultat }
  Writeln('La somme des factorielles de 1 à ', N, ' est : ', somme);
end.

```



☞ *Programme pour calculer la somme des factorielles de la fonction S_2 , c'est-à-dire la somme :*

$$S_2 = 1 - \frac{X^2}{3!} + \frac{X^4}{5!} - \frac{X^6}{7!} + \dots \pm \frac{X^{2n}}{(2n+1)!}$$

Cette expression est une série alternée :

- ❖ Le numérateur est la puissance de X (par exemple X^2, X^4, X^6 , etc.).
- ❖ Le dénominateur est la factorielle de nombres impairs successifs (par exemple $3!, 5!, 7!$, etc.).
- ❖ Les signes alternent entre positif et négatif, à savoir : +, - et ainsi de suite :

Chaque terme est de la forme :

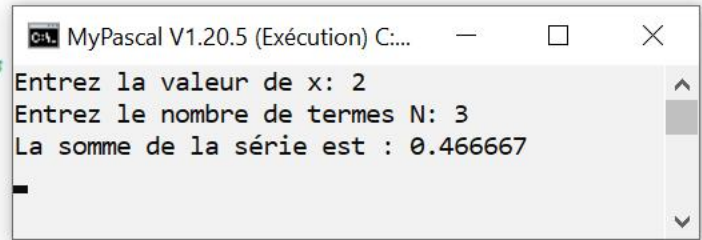
$$S_2 = \sum_{i=0}^{N-1} (-1)^i \frac{x^{2i}}{(2i+1)!}$$

Où i est l'indice du terme (de 0 à N-1).

$(-1)^i$: Permet d'alterner les signes de chaque terme de la série.

☞ *Compilation et exécution du programme pour $x=2$ et $N=3$*

```
program Somme_S_2;
var
  x: Real;
  N, i, signe: Integer;
  somme, fact, P: Real;
begin
  // Demander à l'utilisateur la valeur de x et du nombre de termes
  Write('Entrez la valeur de x: ');
  ReadLn(x);
  Write('Entrez le nombre de termes N: ');
  ReadLn(N);
  // Initialisation de la somme à 0
  somme := 0.0; signe := -1; fact := 1.0; // 1! pour le premier terme
  // Calcul de la somme de la série
  for i := 0 to N - 1 do
  begin
    signe := (-1) * signe;
    // Calcul de  $x^{(2i)}$  en utilisant la fonction Power
    P := signe * exp((i*2)*ln(x));
    somme := somme + P / fact;
    // Calcul de  $(2i+1)!$ 
    fact := fact * (2 * i + 2) * (2 * i + 3);
  end;
  // Afficher la somme de la série
  WriteLn('La somme de la série est : ', somme:0:6);
end.
```



```
MyPascal V1.20.5 (Exécution) C:...
Entrez la valeur de x: 2
Entrez le nombre de termes N: 3
La somme de la série est : 0.466667
```

Solution de l'exercice N°03 :

Description du problème :

- On commence par lire les valeurs A, B et N, où A et B sont des entiers positifs, et N est le nombre de valeurs à entrer.
- Ensuite, on lit ces N valeurs, une par une.
- On vérifie si chaque valeur est paire ou impaire :
 - Si elle est paire, on l'ajoute à une variable somme_pairs.
 - Si elle est impaire, on la multiplie à une variable produite_impairs.
- À la fin, on affiche la somme des pairs et le produit des impairs.

☞ Programme en PASCAL

```
Program SommeProduitPairsImpairs;
var
  A, B, N, i, valeur : integer;
  somme_pairs, produit_impairs : longint; { longint compris entre  $-2^{31}$  et  $2^{31}-1$  }
Begin
  { Entrée des bornes A et B, ainsi que du nombre N de valeurs }
  Write('Entrez la valeur de A (A < B) : ');
  Readln(A);
  Write('Entrez la valeur de B (A < B) : ');
  Readln(B);
  Write('Entrez le nombre N de valeurs à entrer : ');
  Readln(N);

  { Initialisation des variables }
  somme_pairs := 0;
  produit_impairs := 1; { On initialise à 1 car c'est le neutre pour la multiplication }
  { Boucle pour entrer les N valeurs }
  for i := 1 to N do
    begin
      Write('Entrez la ', i, 'ème valeur (comprise entre ', A, ' et ', B, ') : ');
      Readln(valeur);
      { Vérification si la valeur est dans l'intervalle [A, B] }
      if (valeur < A) or (valeur > B) then
        begin
          WriteLn('Erreur : La valeur doit être comprise entre ', A, ' et ', B, '.');
          continue; { On passe à l'itération suivante si la valeur est hors de l'intervalle }
        end;
      { Traitement des nombres pairs et impairs }
      if valeur mod 2 = 0 then
        somme_pairs := somme_pairs + valeur { La valeur est paire, on l'ajoute à la somme }
      else
        produit_impairs := produit_impairs * valeur; { La valeur est impaire, on la multiplie au
produit }
    end;
    { Affichage des résultats }
    WriteLn('La somme des valeurs paires est : ', somme_pairs);
    WriteLn('Le produit des valeurs impaires est : ', produit_impairs);
  end.
```

☞ Compilation et exécution du programme pour A=5, B= 15 et N=5 :

```

program SommeProduitPairsImpairs;
var
  A, B, N, i, valeur: integer;
  somme_pairs, produit_impairs: longint;
begin
  { Entrée des bornes A et B, ainsi que du nombre N de valeurs }
  Write('Entrez la valeur de A (A < B) : ');
  Readln(A);
  Write('Entrez la valeur de B (A < B) : ');
  Readln(B);
  Write('Entrez le nombre N de valeurs à entrer : ');
  Readln(N);
  { Initialisation des variables }
  somme_pairs := 0;
  produit_impairs := 1; { On initialise à 1 car c'est le neutre pour la multiplication }
  { Boucle pour entrer les N valeurs }
  for i := 1 to N do
  begin
    Write('Entrez la ', i, 'ème valeur (comprise entre ', A, ' et ', B, ') : ');
    Readln(valeur);
    { Vérification si la valeur est dans l'intervalle [A, B] }
    if (valeur < A) or (valeur > B) then
    begin
      Writeln('Erreur : La valeur doit être comprise entre ', A, ' et ', B, '.');
      continue; { On passe à l'itération suivante si la valeur est hors de l'intervalle }
    end;
    { Traitement des nombres pairs et impairs }
    if valeur mod 2 = 0 then
    | somme_pairs := somme_pairs + valeur { La valeur est paire, on l'ajoute à la somme }
    else
    | produit_impairs := produit_impairs * valeur; { La valeur est impaire, on la multiplie au produit }
    end;
  { Affichage des résultats }
  Writeln('La somme des valeurs paires est : ', somme_pairs);
  Writeln('Le produit des valeurs impaires est : ', produit_impairs);
end.

```

```

MyPascal V1.20.5 (Exécution) C:\Users\Maison\...
Entrez la valeur de A (A < B) : 5
Entrez la valeur de B (A < B) : 15
Entrez le nombre N de valeurs à entrer : 5
Entrez la 1ème valeur (comprise entre 5 et 15) : 6
Entrez la 2ème valeur (comprise entre 5 et 15) : 7
Entrez la 3ème valeur (comprise entre 5 et 15) : 8
Entrez la 4ème valeur (comprise entre 5 et 15) : 9
Entrez la 5ème valeur (comprise entre 5 et 15) : 10
La somme des valeurs paires est : 24
Le produit des valeurs impaires est : 63

```

Solution de l'exercice 04 :

Description du problème :

☞ Entrées :

La valeur maximale de la masse N que l'utilisateur entre. Cette valeur doit être un entier supérieur ou égal à 2.

☞ Traitement :

Le calcul de l'**accélération** « a » pour chaque masse m de **2kg** jusqu'à N , avec un pas de **3kg** à chaque itération. L'accélération a est donnée par la formule : $a = m \times g$ où $g = 9,8 \text{ m/s}^2$ est la constante de la gravité.

☞ Sorties :

Pour chaque masse m , on affiche l'accélération a correspondante, avec deux décimales.

Solution avec la boucle *while*

Nous remarquons, pour contrôler plus précisément l'incrémentation à chaque itération (c'est-à-dire, pour augmenter m de 3 à chaque fois), la boucle *while* est plus simple à utiliser. Il est spécifié que la masse m doit commencer à 2Kg, puis augmenter par un *pas de 3* à chaque itération (2, 5, 8, 11, 14, ...), jusqu'à atteindre la valeur N .

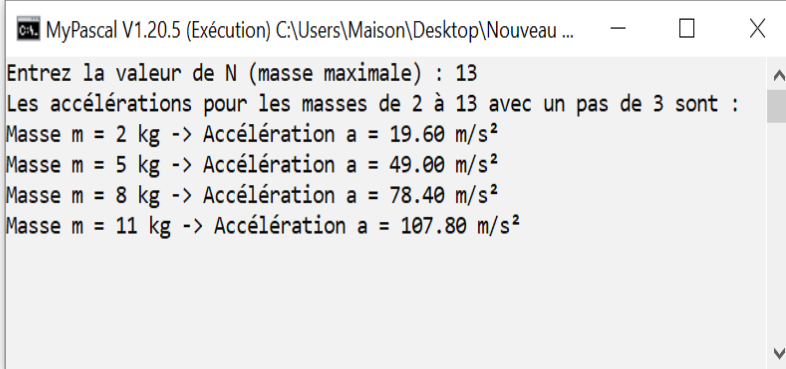
L'approche consiste à utiliser une **boucle *while***. Ce qui nous permet de contrôler complètement l'itération en incrémentant la variable m de 3 à chaque itération :

NB :

La boucle *for* classique de Pascal **n'incrmente la variable de contrôle que de 1.**

☞ *Compilation et exécution du programme pour N=13 :*

```
program CalculAccelerations;
var
  a: real;    { Accélération }
  N, m: integer; { N : masse maximale, m : masse courante }
const
  g = 9.8;    { Gravité en m/s² }
begin
  {****_****Entrées****_****}
  Write('Entrez la valeur de N (masse maximale) : ');
  Readln(N);
  {****_****Traitement****_****}
  m := 2; { On commence avec une masse de 2 kg }
  Writeln('Les accélérations pour les masses de 2 à ', N, ' avec un pas de 3 sont :');
  while m <= N do
  begin
    a := m * g; { Calcul de l'accélération a = m * g }
    Writeln('Masse m = ', m, ' kg -> Accélération a = ', a:0:2, ' m/s²');
    m := m + 3; { On incrémente la masse de 3 à chaque itération }
  end;
end.
```



```
MyPascal V1.20.5 (Exécution) C:\Users\Maison\Desktop\Nouveau ...
Entrez la valeur de N (masse maximale) : 13
Les accélérations pour les masses de 2 à 13 avec un pas de 3 sont :
Masse m = 2 kg -> Accélération a = 19.60 m/s²
Masse m = 5 kg -> Accélération a = 49.00 m/s²
Masse m = 8 kg -> Accélération a = 78.40 m/s²
Masse m = 11 kg -> Accélération a = 107.80 m/s²
```