

Sommaire

Chapitre 1 : Sous-Programmes - Procédures et Fonctions (Partie 2).....	2
1.6. Exercices d'Application (suite).....	2
1.6.2. Calcul de combinaisons (avec deux fonctions).....	2
1.6.3. Calcul de puissance x^n	3
1.6.4. Transformer une fonction à une procédure.....	5
1.6.5. Calcul de PGCD et PPCM.....	6
1.7. Fonctions/Procédures récursives.....	9
1.7.1. Notion de récursivité.....	9
1.7.2. Exemple 1 : Factoriel.....	9
1.7.3. Exemple 2 : Puissance.....	10
1.7.4. Exercices.....	11
a) PGCD.....	11

Chapitre 1 : Sous-Programmes - Procédures et Fonctions (Partie 2)

1.6. Exercices d'Application (suite)

1.6.2. Calcul de combinaisons (avec deux fonctions)

Une autre solution pour calculer la combinaison et de déclarer deux fonctions : la fonction factoriel et la fonction combinaison :

$$C_n^k = \frac{n!}{k! \times (n-k)!} \quad \text{si } n \geq k \geq 1$$

$$C_n^k = 0 \quad \text{si } n < k$$

```

Program combinaisonAvec2Fonctions;
uses wincrt;
var
    n, k, c : integer;
Function factoriel(n:integer):integer;
var
    j, f:integer;
Begin
    f:=1;
    for j:=1 to n do
        f:= f*j;
    factoriel := f;
End;

Function combinaison(n:integer ; k:integer):integer;
var
    c, fn, jk, jnk:integer;
Begin
    c:=0;

    if k<=n then
        begin
            fn := factoriel(n);
            fk := factoriel(k);
            fnk := factoriel(n-k);
            c:= fn / (fk * fnk);
        end;

    combinaison:=c;
End;

BEGIN {Début du programme principal}
    read(n, k);
    c := combinaison (n, k);
    write('Combinaison n k = ', c);
END. {Fin du programme principal}
  
```

Diagram annotations:

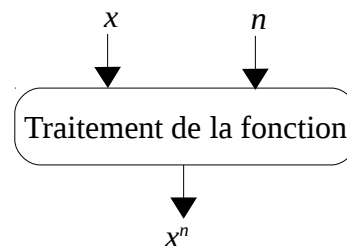
- Variables globales du programme**: points to the `var n, k, c : integer;` declaration.
- Paramètre formel**: points to the `n:integer` parameter in the `factoriel` function signature.
- Variables locales de la fonction *factoriel***: points to the `var j, f:integer;` declaration inside the `factoriel` function.
- Dernière instruction de la fonction**: points to the `factoriel := f;` statement at the end of the `factoriel` function.
- Paramètres formels (séparés par point-virgule)**: points to the `n:integer ; k:integer` parameters in the `combinaison` function signature.
- Variables locales de la fonction *combinaison***: points to the `var c, fn, jk, jnk:integer;` declaration inside the `combinaison` function.
- Trois appels à la fonction factoriel avec les paramètres respectifs : n, k et n-k.**: points to the three calls `fn := factoriel(n);`, `fk := factoriel(k);`, and `fnk := factoriel(n-k);` inside the `combinaison` function.
- Dernière instruction de la fonction**: points to the `combinaison:=c;` statement at the end of the `combinaison` function.
- Appels à la fonction combinaison, avec les paramètres effectifs: n et k. (Séparés par des virgules)**: points to the call `c := combinaison (n, k);` in the main program.

1.6.3. Calcul de puissance x^n

Écrire une fonction PASCAL permettant de calculer la valeur de x^n tels-que x est un nombre réel et n un nombre entier positif.

Pour faciliter l'analyse du problème, on considère la fonction comme une boîte noire avec des entrées et des sorties. Les entrées sont le paramètres en entrées (transmission par valeur), et les sorties sont la valeur de la fonction et éventuellement les paramètres en sorties (transmission par variable).

La fonction puissance peut être vue comme une boîte possédant deux entrées x (nombre réel) et n (nombre entier positif) et retourne un résultat réel égale $p=x^n$.



Pour le traitement de la fonction, nous avons la propriété mathématique suivant :

$$p = x^n = x \times x \times \dots \times x \text{ (} n \text{ facteurs)} = \prod_{i=1}^n x$$

On peut remplacer le signe de produit par une boucle pour avec un compteur allant de 1 jusqu'à n comme suit :

```

p:=1;
Pour i<-1 à n faire
    p ← p * x
FinPour
  
```

La fonction puissance s'écrit comme suit :

```

1 Function puissance (x:real ; n:integer):real ;
2
3 var
4     i:integer ;
5     p:real;
6 Begin
7     p:=1;
8     For i:=1 to n do
9         p := p*x;
10
11     puissance:=p;
12 End;
  
```

Dernière instruction de la fonction

N.B. : La fonction précédente doit être déclarée au sein d'un programme pour qu'elle puisse être appelée et utilisée comme illustrée dans l'exemple qui suit.

Écrire un programme Pascal permettant de calculer la somme S définie comme suit :

$$S = \sum_{i=0}^n \frac{x^i}{i!} = 1 + \frac{x}{1} + \frac{x^2}{2} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots + \frac{x^n}{n!}$$

En utilisant les deux fonctions puissances et factoriel.

```

01 Program SommePowFact;
02 uses wincrt;
03   var
04     n, i:integer;
05     x, S : real;
06
07   Function factoriel(n:integer):integer;
08   var
09     i,f:integer;
10   Begin
11     f:=1;
12     for i:=1 to n do
13       f := f*i;
14     factoriel:=f;
15   End;
16
17   Function puissance (x:real;n:integer):real;
18   var
19     i:integer ;
20     p:real;
21   Begin
22     p:=1;
23     for i:=1 to n do
24       p := p*x;
25     puissance:=p;
26   End;
27
28 BEGIN {début du Programme principal}
29   Write ('Donnez une valeur réelle x : '); read(x);
30   Write ('Donnez une valeur entière n : '); read(n);
31   S:=0;
32   for i:=0 to n do
33     S:= S + puissance(x,i)/factoriel(i);
34
35   Write ('La somme S = ',S:6:2);
36 END. {fin du programme principal}

```

La partie déclaration du programme qui contient quatre variables (2 entiers et 2 réels), et deux fonctions.

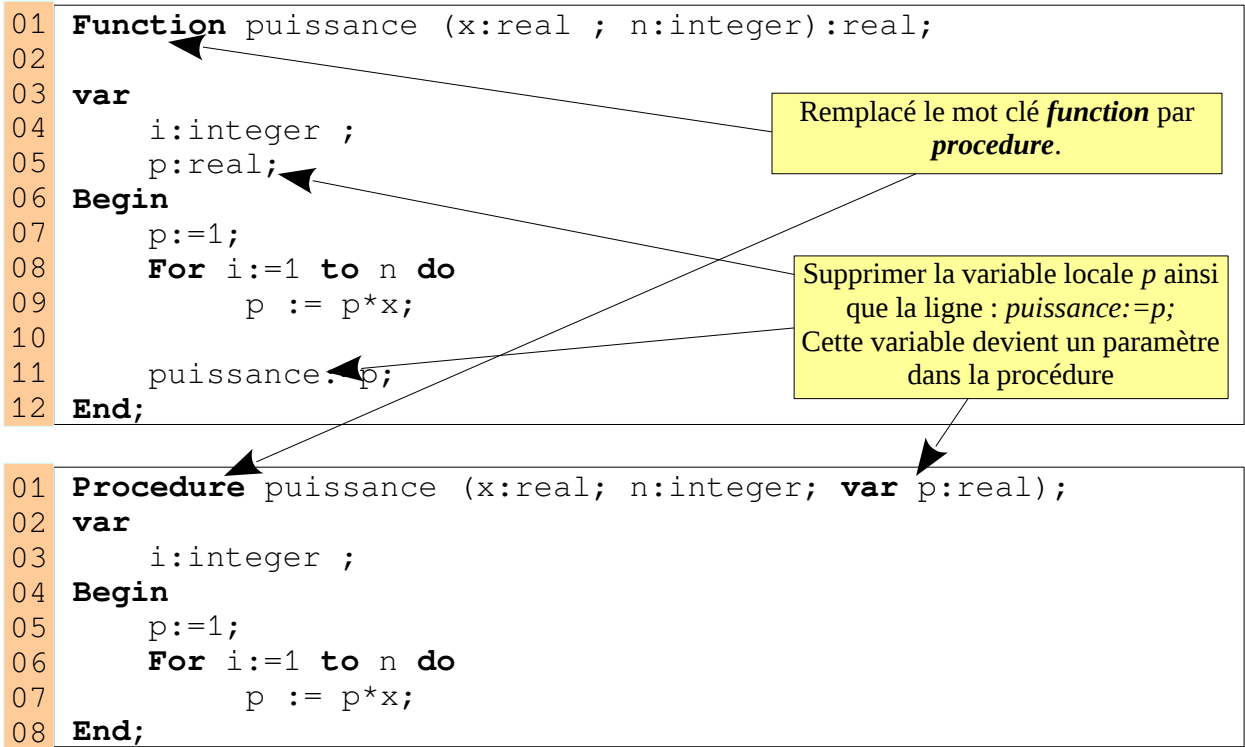
Le corps du programme principal

Ainsi, toute fonction doit être déclarée (dans la partie de déclaration du programme) avant d'être appelée (utilisée).

1.6.4. Transformer une fonction à une procédure

Pour transformer une fonction à une procédures, on ajoute un paramètre additionnel à cette dernière (procédure). Si la fonction contient n paramètres alors le nombre de paramètres de la procédure sera $n+1$. Ce paramètre sera précédé par le mot clé **VAR** pour indiquer une transmission de paramètre par variables (paramètre de sortie).

Pour illustrer les étapes à suivre pour transformer une fonction à une procédure, on utilise l'exemple suivant :



Les étapes à suivre sont :

- i-* Remplacer le mot clé **function** par le mot clé **procedure** ;
- ii-* Ajouter un paramètre additionnel à la procédure avec une transmission par variable (utiliser le mot clé **VAR**). Ce paramètre remplace le résultat de la fonction ;
- iii-* Supprimer le type de la fonction ;
- iv-* Enlever la variable locale qui contient le résultat (c'est la variable affectée à la fonction à la fin de la même fonction) ;
- v-* la variable de résultat de la fonction et le même paramètre additionnel ajouté à la procédure.
- vi-* Enlever la dernière ligne de la fonction.

Bien évidemment, l'appel à une fonction est différent de l'appel à la procédure. Par exemple :

<i>Dans le cas de la fonction</i>	<i>Dans le cas de la procédure</i>
<code>x := puissance(5, 3)</code>	<code>puissance(5, 3, x);</code>
<code>z := puissance(x+y, 10);</code>	<code>puissance(x+y, 10, z)</code>

Dans le cas de la fonction, on réalise une affectation de la fonction vers une variable, par contre, dans le cas de la procédure, la variable devant contenir le résultat sera utilisée comme le paramètre effectif du paramètre transmis par variable.

- **Transformer la fonction factoriel à une procédure :**

```

01 Function factoriel (n:integer):integer;
02 var
03     i, f:integer ;
04 Begin
05     f:=1;
06     for i:=1 to n do
07         f:= f*i;
08     factoriel:=f;
09 End;

```

```

01 procedure factoriel (n:integer; var f:integer);
02 var
03     i:integer ;
04 Begin
05     f:=1;
06     for i:=1 to n do
07         f:= f*i;
08 End;

```

1.6.5. Calcul de PGCD et PPCM

Écrire une procédure PASCAL permettant de calculer le plus grand commun diviseur (PGCD) et le plus petit commun multiple (PPCM) de deux nombres entiers a et b .

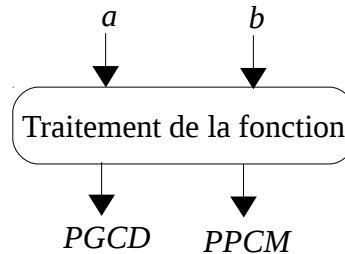
Le PGCD de deux nombres a et b se calcule en suivant les étapes suivantes :

- 1- Mettre a dans $a1$ et mettre b dans $b1$
- 2- si $a1$ est égale à $b1$ donc aller à l'étape 7
- 3- s'ils sont différents, on aura deux cas :
 - 4- Si $a1 > b1$ donc on fait $a1$ reçoit $a1-b1$
 - 5- Si $b1 > a1$ donc on fait $b1$ reçoit $b1-a1$
- 6- aller à l'étape 2

7- $PGCD = a1$ et afficher le $PGCD$

8- $PPCM = x*y / PGCD$ et afficher le $PPCM$

Pour cette procédure, on a deux paramètres d'entrée (passage par valeur) : deux nombre entier a et b . Et deux paramètre de sortie (passage par variable) : $PPCM$ et $PGCD$.



Donc, les deux paramètres a et b sont des paramètres d'entrée (passage par valeur : sans le mot clé **VAR**). Par contre, les deux paramètre $PGCD$ et $PPCM$ sont des paramètres de sorties (passage par variable : on les précède par le mot clé **VAR**)

```

01 procedure PGCD_PPCM(a,b:integer; var PGCD,PPCM:integer);
02 var
03     a1, b1 : integer ;
04 Begin
05     a1:=a; b1:=b;
06
07     if (a1=0) then
08         a1:=b1
09     else
10         if (b1 = 0) then
11             b1:=a1 ;
12
13
14     while (a1 <> b1) do
15         begin
16             if (a1 > b1) then
17                 a1 := a1 - b1
18             else
19                 b1 := b1 - a1;
20         end;
21
22     PGCD := a1;
23     PPCM := (a*b) div PGCD ;
24 End;

```

Le programme suivant illustre l'utilisation de la fonction `PGCD_PPCM` :

```

program Calcul_PGCD_PPCM;
uses wincrt;

var
    n, m : integer;
    pg, pp : integer;

procedure PGCD_PPCM(a,b:integer; var PGCD,PPCM:integer);
var
    a1, b1:integer;
Begin {Début de la procédure}
    a1:=a; b1:=b;

    if (a1=0) then
        a1:=b1
    else
        if (b1 = 0) then
            b1:=a1;

    while (a1 <> b1) do
        begin
            if (a1 > b1) then
                a1 := a1 - b1
            else
                b1 := b1 - a1;
        end;

    PGCD := a1;
    PPCM := (a*b) div PGCD;
End;

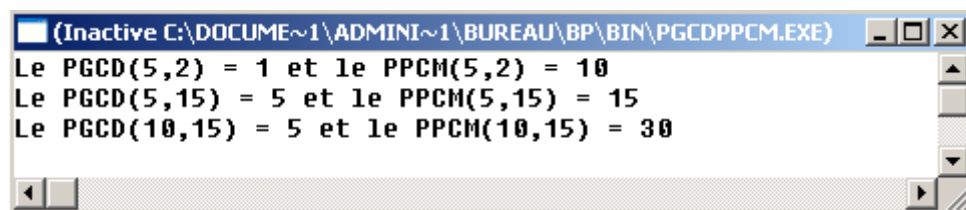
BEGIN {Début du programme principal}
    n:=5; m:=2;
    PGCD_PPCM(n, m, pg, pp);
    Writeln ('Le PGCD(', n, ', ', m, ') = ', pg, ' et le PPCM(', n, ', ', m, ') = ', pp);

    n:=5; m:=15;
    PGCD_PPCM(n, m, pg, pp);
    Writeln ('Le PGCD(', n, ', ', m, ') = ', pg, ' et le PPCM(', n, ', ', m, ') = ', pp);

    n:=10; m:=15;
    PGCD_PPCM(n, m, pg, pp);
    Writeln ('Le PGCD(', n, ', ', m, ') = ', pg, ' et le PPCM(', n, ', ', m, ') = ', pp);
END. {Fin du programme principal}

```

Le résultat sera affiché comme suit :



```

(Inactive C:\DOCUME~1\ADMINI~1\BUREAU\BP\BIN\PGCDPPCM.EXE)
Le PGCD(5,2) = 1 et le PPCM(5,2) = 10
Le PGCD(5,15) = 5 et le PPCM(5,15) = 15
Le PGCD(10,15) = 5 et le PPCM(10,15) = 30

```

1.7. Fonctions/Procédures récursives

1.7.1. Notion de récursivité

Un sous-programme récursif est un sous-programme qui fait appel à lui même d'une manière directe ou indirecte. On utilise les sous-programmes récursifs en général soit pour résoudre les problèmes mathématiques avec récurrences (comme le factoriel, la puissance, *etc.*) ou bien pour les structures de données récurrentes (comme les listes, les arbres, *etc.*).

Une procédure ou une fonction récursive permet de réaliser le même traitement plusieurs fois, donc, (dans quelques cas), on peut les utiliser pour remplacer les boucles (**for**, **while** et **repeat**).

1.7.2. Exemple 1 : Factoriel

Pour calculer le factoriel de n (*nombre entier positifs*), nous savons que :

$$n! = \begin{cases} 1 & \text{si } n=0 \\ 1 \times 2 \times \dots \times n & \text{si } n>0 \end{cases}$$

$$\text{Ce qui fait : } n! = \begin{cases} 1 & \text{si } n=0 \\ 1 \times 2 \times \dots \times (n-1) \times n & \text{si } n>0 \end{cases}$$

$$\text{Donc : } n! = \begin{cases} 1 & \text{si } n=0 \\ (n-1)! \times n & \text{si } n>0 \end{cases}$$

Donc le factoriel de n égale 1 si $n=0$, ou bien égale à factoriel de $(n-1)$ multiplié par n . Donc le factoriel de n dépend du factoriel de $(n-1)$, et le factoriel de $(n-1)$ dépend du factoriel de $(n-2)$ et ainsi de suite (jusqu'à arriver à $n=1$ ou bien $n=0$).

$$\begin{aligned} n! &= (n-1)! \times n \\ (n-1)! &= (n-2)! \times (n-1) \\ (n-2)! &= (n-3)! \times (n-2) \\ &\vdots \\ (1)! &= (0)! \times (0)! \\ (0)! &= 1 \end{aligned}$$

D'après les formules précédentes, chaque factoriel d'un nombre dépend du factoriel du nombre précédent. Jusqu'à arrive à 0, où par définition $0! = 1$.

Comment réaliser une fonction Pascal permettant d'exploiter cette propriété. Le code suivant montre les deux fonction factoriel (la première non récursive et la seconde récursive) :

```

01 Function factoriel (n:integer):integer;
02 var
03     i, f:integer ;
04 Begin
05     f:=1;
06     for i:=1 to n do
07         f:= f*i;
08     factoriel:=f;
09 End;

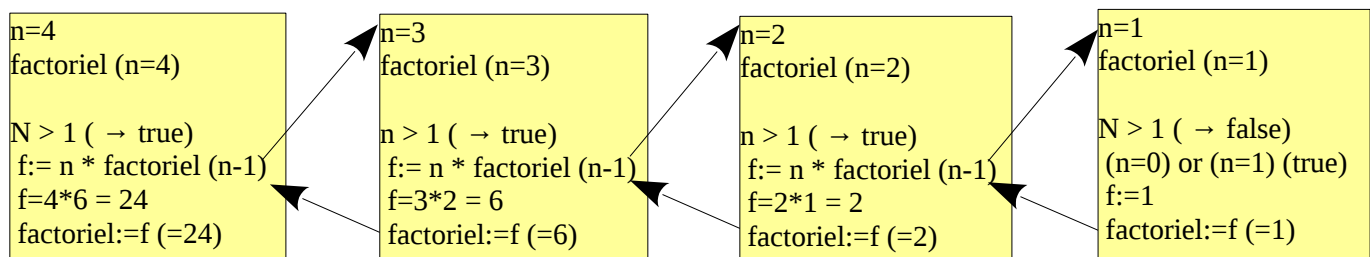
```

```

01 Function factoriel (n:integer):integer;
02 var
03     f:integer;
04 Begin
05     if (n > 1) then
06         f := n * factoriel(n-1)
07     else
08         if (n = 0) or (n = 1) then
09             f:=1;
10     factoriel := f;
11 End;
12

```

On déroule la fonction factoriel pour n = 4 :



1.7.3. Exemple 2 : Puissance

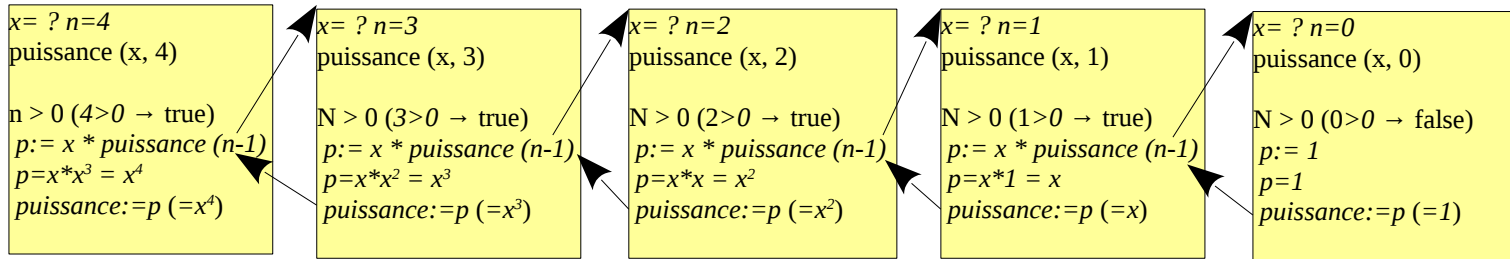
Calcul de la puissance x^n . On a : $x^n = x^{(n-1)} * x$ et on a aussi : $x^0 = 1$.

```

01 Function puissance (x:real; n:integer):real;
02 var
03     p:real;
04 Begin
05     if (n > 0) then
06         p := x * puissance(x, n-1)
07     else
08         if (n = 0) then
09             p:=1;
10     puissance := p;
11 End;

```

On déroule la fonction puissance pour x quelconque et $n=4$:



Remarques :

- Une procédure ou fonction récursive permet de réaliser un traitement répétitif. Cette répétition est représentée par le fait que le sous-programme appelle lui-même.
- Il y a toujours un test (condition) permettant d'éviter une récurrence infinie (traitement infini). De cette façon, on est sûr qu'à une étape donnée, le sous-programme n'appelle plus lui-même et retourne une valeur connue. Ce test ou cette condition s'appelle l'*issue de secours* du sous-programme récursif.
- L'issue de secours de la fonction factoriel est l'expression booléenne : $(n=0) \text{ or } (n=1)$. Et l'issue de secours de la fonction puissance est l'expression booléenne : $n=0$.

1.7.4. Exercices

Voici une liste d'exercice sur les sous-programmes récursifs :

a) PGCD

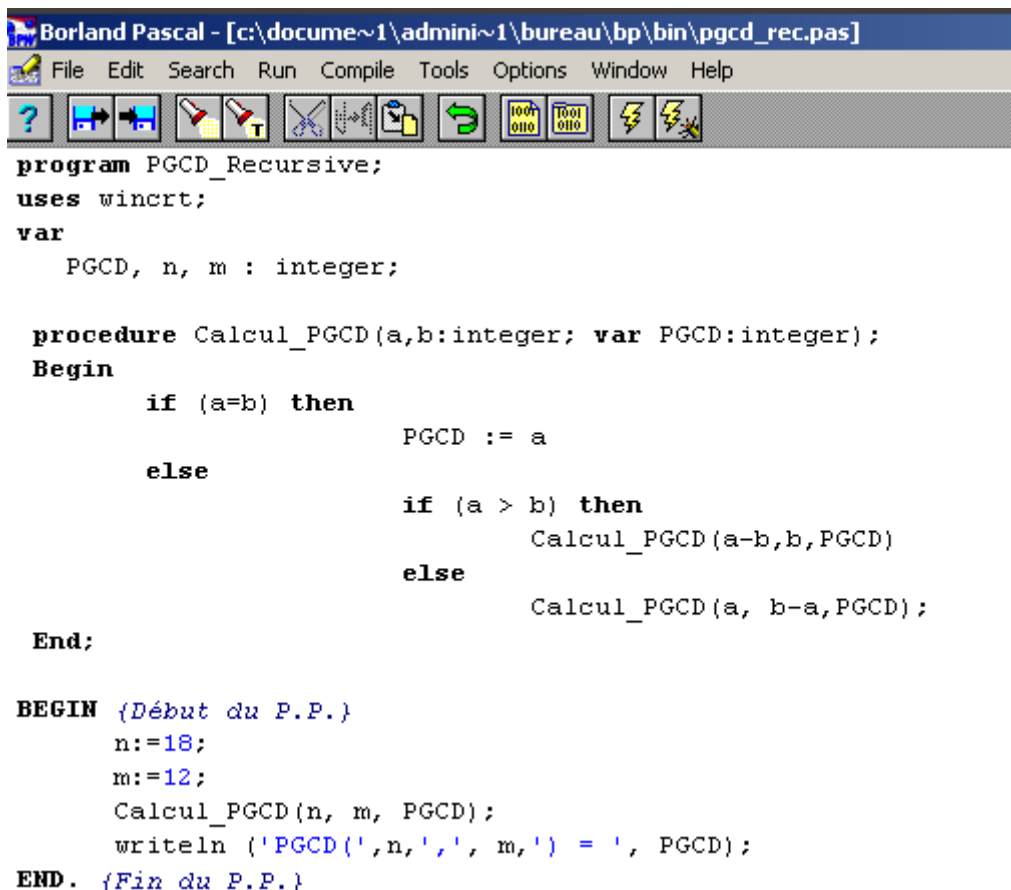
Écrire une procédure récursive permettant de calculer le plus grand commun diviseur commun entre deux nombres entiers strictement positifs. En applique la méthode indiquée dans la section [calcul de PGCD et PPCM](#).

Solution :

```

01 procedure Calcul_PGCD(a,b:integer; var PGCD:integer);
02 Begin
03     if (a=b) then
04         PGCD := a
05     else
06         if (a > b) then
07             Calcul_PGCD(a-b, b ,PGCD)
08         else
09             Calcul_PGCD(a, b-a ,PGCD);
10 End;
  
```

La capture d'écran suivante montre un programme PASCAL utilisant la procédure précédente :



```

program PGCD_Recursive;
uses wincrt;
var
    PGCD, n, m : integer;

procedure Calcul_PGCD(a,b:integer; var PGCD:integer);
Begin
    if (a=b) then
        PGCD := a
    else
        if (a > b) then
            Calcul_PGCD(a-b,b,PGCD)
        else
            Calcul_PGCD(a, b-a,PGCD);
    End;

BEGIN {Début du P.P.}
    n:=18;
    m:=12;
    Calcul_PGCD(n, m, PGCD);
    writeln ('PGCD(' , n, ', ' , m, ') = ' , PGCD);
END. {Fin du P.P.}

```

Dérouler le programme précédent.

a) Racine carrée par la méthode de Newton

Écrire une fonction non récursive permettant de calculer une valeur approximative de la racine carrée d'un nombre réel X en utilisant la méthode de newton. Les étapes de cette méthode est comme suit :

- 1- Soit X un nombre réel positif non nul pour le quel on veut calculer la racine carée
- 2- Soit Epsilon un nombre réel positif qui tend vers zéro (très petit nombre réel positif)
- 3- Soit $R2$ une valeur quelconque (on peut mettre initialement $R2 = X$).
- 4- $R1 \leftarrow R2$
- 5- Calculer $R2 = (R1 + X/R1) / 2$
- 6- Si $|R1 - R2| > \text{Epsilon}$ aller à l'étape 4
- 7- Sinon : donc la racine carrée est $R2$.

```
01 Function racine_carree (x:real):real;  
02   const  
03     epsilon = 0.0000001;  
04   var  
05     R1, R2 : real;  
06 Begin  
07   R2:=x;  
08   repeat  
09     R1:=R2;  
10     R2:=(R1 + X/R1) / 2;  
11   until abs (R1-R2)<=epsilon;  
12   racine_carree:=R2;  
13 End;
```

La version récursive de cette fonction est :

```
01 Function racine_carree (x:real ; R1:real):real;  
02   const  
03     epsilon = 0.0000001;  
04   var  
05     R2 : real;  
06 Begin  
07   R2:=(R1 + X/R1) / 2;  
08   if abs(R1-R2) <= epsilon then  
09     racine_carree:=R2  
10   else  
11     racine_carree := racine_carree(x,R2);  
12 End;
```