Solution de la série de TP N° 02 (Les Sous-Programmes)

Exercices 01 et 02 + Exercices Supplémentaires

Sommaire

Cours Elearning :
https://elearning.univ-bejaia.dz/course/view.php?id=7944

Page facebook :
https://www.facebook.com/InitiationAlgoProgrammation/

La chaîne Youtube :
https://www.youtube.com/channel/UCdS5cy1hCze0fYr3dQoJVmg

Réalisé par : OUZEGGANE Redouane
rouzeggane@gmail.com - redouane.ouzeggane@univ-bejaia.dz

PS :

Nous tenons à s'excuser des erreurs qui peuvent apparaître dans ce document.
SI vous trouverez des erreurs, dans l'explication ou bien dans un algorithme ou programme, veuillez les rapporter sur :

- la page facebook : https://www.facebook.com/InitiationAlgoProgrammation/
- la chaîne youtube : https://www.youtube.com/channel/UCdS5cy1hCze0fYr3dQoJVmg

Série 2 : Les Sous-Programmes : Procédure & Fonctions

Exercice 01 : Concepts de base : Procédure – Paramètres formels & effectifs

Exécuter le programme suivant :

La partie déclaration
On a déclaré trois variables globales et deux procédures

Le Corps du Programme
La partie instructions

```
01 Program Exo_1;
02 Uses wincrt ;
03 var
04    a, b, c : real;{Variables Globales du programme}
05
06 Procedure Proc1(x:real; y:real; var s:real) ;
07  Begin
08    s := x / y ;
09  End;
10
11 Procedure Proc2(x:real; y:real; s:real);
12  Begin
13    s := x / y;
14  End;
15
16 BEGIN {Début du Programme Principal}
17    a:=10; b:=5; c:=0;
18    Proc1(a, b, c);
19    Writeln('La division est : ', c);
20
21    a:=10; b:=5; c:=0;
22    Proc2(a, b, c);
23    Writeln('La division est : ', c);
24 END. {Fin du Programme Principal}
```

L'appel à la procédure *Proc1*, en transmettant les paramètres : a, b et c

L'appel à la procédure *Proc2*, en transmettant les paramètres : a, b et c

- C'est quoi la différence entre les deux procédures *Proc1* et *Proc2* ?

- Quels sont les paramètres à passage par valeur et ceux à passage par variable ?

- Quels sont les paramètres formels des deux procédures ?

- Et quels sont les paramètres effectifs ?

Solution

1- La différence entre les deux procédures Proc1 et Proc2

La différence réside dans le troisième paramètre **s** . Dans *Proc1*, le paramètre **s** est défini en utilisant le mot clé **var**. Par contre, dans *Proc2* il est défini sans utilisation du mot clé **var**.

Ceci aura un effet dans l'appel au deux procédures, c'est quoi cette effet ? Ça sera quoi la différence entres les deux appels ? (voir les réponses suivantes …)

2- Les paramètres à passage par valeur et ceux à passage par variable

Pour la procédure *Proc1* :

- Passage par Valeur : les paramètres x (réel), y (réel)
- Passage par Variable: le paramètre S (réel)

Pour la procédure *Proc2* :

- Passage par Valeur : les paramètres x (réel) et y (réel) et s (réel)
- Passage par Variable : aucun paramètre

La définition de type de passage de paramètres (par valeur ou par variable) se fait dans la partie déclaration du programme principal, où on déclare les procédures ou les fonctions. Dans le programme précédent, la partie déclaration se situe entre les lignes *03* et *14*.

3, 4- Les paramètre formels et les paramètres effectifs :

Pour la procédure *Proc1* :

- Paramètres formels : x, y et s
- Paramètres effectifs : a, b et c (dans l'appel de la ligne n° 18)

Pour la procédure *Proc2* :

- Paramètres formels : x, y et s
- Paramètres effectifs : a, b et c (dans l'appel de la ligne n° 22)

C'est quoi les paramètres formels ? Les paramètres formels sont les paramètres utilisés pour déclarer un sous-programme (procédure ou fonction). Donc pour chercher les paramètres formels d'un sous programme, il faut voir dans la partie déclaration du programme principal.

Et pour les ***paramètres effectifs***, il faut regarder dans la partie du corps du programme principal (partie instruction), pour d'éventuels appels à un sous programme. Pour chaque appel à une procédure ou à une fonction, on doit indiquer les paramètres effectifs.

Par exemple, dans la ligne n°22, nous avons l'instruction : *Proc2(a, b, c) ;* ce qui signifie, l'appel à la procédure *Proc2* avec les paramètres effectifs a, b et c. Dans ce cas là, la valeur de *a* sera transmise au paramètre formel *x*, celle de *b* sera transmise au paramètre formel y et la valeur de c sera transmise à s.

Remarques :

- Les paramètres effectifs sont liés à l'appel. Si une fonction ou procédure n'est pas appelée, donc elle n'a pas de paramètre effectif.

- Si un sous programme (procédure ou fonction) est appelé plusieurs fois dans le programme principal, nous devons indiquer les paramètres effectifs par appel :

les paramètres effectifs pour le premier appel sont : ….
les paramètres effectifs pour le deuxième appel sont : …
etc.

5- Déroulement du programme

Le déroulement est d'exécuter manuellement un algorithme (ou un programme) et visualiser le changement des valeurs de variables. À la fin de déroulement, nous aurons les valeurs des variables de sortie qui représentent la solution du problème.

La méthode que nous utilisons pour dérouler un problème consiste à utiliser un tableau de deux colonnes : Instructions et Variables. La colonne variables est divisée en sous colonnes : les variables de l'algorithme, comme illustré ci-dessous :

Instructions	***Variables***			
	V_1	V_2	...	V_n
Instruction 1				
Instruction 2				
….				

Dans cette exercice (et dans tous les exercices de cette série de T.P.), nous avons aussi des sous-programmes définis au sein du programme principal. Donc, il faut séparer entre :

- les variables globales (variable du programme principal)

- les paramètres et les variables locales de chaque sous-programme (fonction ou procédure).

Ainsi, chaque sous-programme aura ses propre variables (Variables locales + paramètres formels).

Le modèle ci-dessous illustre comment concevoir le tableau de de déroulement avec des sous programmes :

Instructions	***Variables***												
	Prog. Principal				***Sous-Prog. 1***				...	***Sous-Prog. n***			
	V_1	V_2	...	V_n	V_1	V_2	...	V_m	...	V_1	V_2	...	V_p
Instruction 1													
Instruction 2													
….													

Le déroulement du programme précédent est détaillé sur la page suivante :

Instructions	***Variables***								
	Programme Principal			***Procédure Proc1***			***Procédure Proc2***		
	a	*b*	*c*	*x*	*y*	*s(var)*	*x*	*y*	*s*
a:=10; b:=5; c:=0;	10	5	0						
Proc1(a, b,c) (L'appel à Proc1)	"	"	"						
=> *La transmission des paramètres* *s:=x/y = 10 / 5 = 2*				10	5	0 **2**			
Fin d'appel : la valeur de s est affectée à C ***La valeur de C sera changée parce que le paramètre formel S est passé par variable.***	10	5	**2**						
write(c) ;			**2**						
a:=10; b:=5; c:=0;	10	5	0						
Proc2(a, b,c) ;*(L'appel à Proc2)*									
=> *La trasmission des paramètres* s:=x/y = 10 / 5 = 2							10	5	0 **2**
Fin d'appel : aucune valeur ne change **(la valeur de s dans Proc1 n'a pas été retournée à la variable globale c)**	10	5	**0**						
write(c) ;	"	"	**0**						

La valeur de S est retournée à c (Passage par variable)

La valeur de S n'est pas retournée à c (Passage par valeur)

Lien du Programme : https://onlinegdb.com/rP8Yy5yMz

6- Remarques

1. Dans le tableau précédant, les zones gris signifie que les **sous-programmes** ne sont pas en exécution et les zone jaune signifie que le **programme principal** a fait l'appel à un sous-programme et attends le retour d'appel (la fin du sous-programme)
2. L'appel *Proc1(a, b, c)* – permet de transmettre les valeurs des paramètres effectifs *a, b* et *c* vers les paramètres formel *x, y* et *s*, par la suite exécuter ses instructions (*s:= x / y;*).
3. L'appel *Proc2(a, b, c)* – permet de transmettre les valeurs des paramètres effectifs *a, b* et *c* vers les paramètres formel *x, y* et *s*. par la suite exécuter ses instructions (*s:= x / y;*).
4. La valeur du paramètre formel *s* de la procédure *Proc1* sera retournée au paramètre effectif *c* (la variable globale *c*). (***La transmission par variable***).
5. La valeur du paramètre formel *s* de la procédure *Proc2* n'a pas été retournée au paramètre effectif *c* (la variable globale *c*). (***La transmission par valeur)***.
6. La transmission (le passage) des paramètres *par valeur* permet de transmettre les valeurs des paramètres effectifs vers les paramètres formels (Paramètres d'entrée).
7. La transmission (le passage) des paramètres *par variable* permet de transmettre les valeurs des paramètres effectifs vers les paramètres formels et, à la fin de l'exécution du sous-programme, de retourner les valeurs des paramètres formel vers les paramètre effectifs correspondant (Paramètres d'entrée et de sortie).

Exercice 02 : Sous-Programme : Fonction – Calcul de combinaisons

```
01 Program CombinaisonNK;
02 Uses wincrt ;
03 Var {Variables Globales du programme}
04     n, k, c : integer;
05     fn, fk, fnk : integer;
06
07 Function fact(n:integer):integer ;
08     Var {Variables locales de la fonction fact}
09         f, i : integer ;
10  Begin
11     F:=1;
12     For i:=1 to n do
13         f:=f*i;
14
15     fact:=f; {Une fonction se termine toujours par une affectation}
16  End;
17
18 BEGIN {Début du Programme Principal}
19     Writeln('Donnez la valeur de n et k :');
20     Read(n, k); {On suppose que n ≥ k}
21
22     fn := fact(n);
23     fk := fact(k);
24     fnk := fact(n-k);
25
26     C:= fn div ( fk * fnk );
27
28     Writeln('Le nombre de combinaisons de k à partir de n = ', c);
29 END. {Fin du Programme Principal}
```

- Dérouler le programme ci-dessus pour ***n = 5*** et ***k = 3***
- Réécrire le programme en transformant le fonction ***fact*** à une procédure ***fact***.
- Dérouler le nouveau programme pur ***n = 5*** et ***k = 4***

Solution

1- Le déroulement du programme pour n=5 et k=3

Instructions	*Variables*									
	Programme Principal						*La fonction fact*			
	n	*k*	*fn*	*fk*	*fnk*	*c*	*n*	*i*	*f*	*fact*
Read(n, k)	**5**	**3**				/				
fn:= fact(n) ;(l'appel à fact avec le paramètre n=5)	**5**	``								
=> La transmission des paramètres → n=5 *f:=1* *for i=1 f:=f*i → f=01*1 = 1* *for i=2 f:=f*i → f=01*2 = 2* *for i=3 f:=f*i → f=02*3 = 6* *for i=4 f:=f*i → f=06*4 = 24* *for i=5 f:=f*i → f=24*5 = 120* *fact := f → **fact = 120***							**5**	/ 1 2 3 4 5	/ 1 1 2 6 24 120	/ **120**
fn:= 120; {fact(n) est remplacé par 120}	``	``	**120**							
fk := fact(k) ; (l'appel à fact avec le paramètre k=3)	**5**	**3**	``	/	/					
=> La transmission des paramètres → n=3 *f:=1* *for i=1 f:=f*i → f=1*1 = 1* *for i=2 f:=f*i → f=1*2 = 2* *for i=3 f:=f*i → f=2*3 = 6* *fact := f → **fact = 6***							**3**	/ 1 2 3	/ 1 1 2 6	/ **6**
fk:= 6; {fact(n) est remplacé par 6}			**120**	**6**	/					
Fnk:= fact(n-k) ;(l'appel à fact avec Par. n-k=2)	``	``	``	``	/					
=> La transmission des paramètres → n=2 *f:=1* *for i=1 f:=f*i → f=1*1 = 1* *for i=2 f:=f*i → f=1*2 = 2* *fact := f → **fact = 2***							**2**	/ 1 2	/ 1 1 2	/ **2**
fnk:= 2; {fact(k) est remplacé par 2}			**120**	**6**	**2**					
*C:= 120 div (6 * 2) → C = 120 div 12 = 10* *Writeln(c);*	``	``	``	``	``	**10** **10**				

Le programme affichera :

```
Le nombre de combinaisons de k à partir de n = 10
```

2- Transformer la fonction fact à une procédure

Pour transformer une fonction à une procédure nous suivons les étapes suivante :

a- Le mot clé ***Function*** devient ***Procedure*** ;

b- On supprime le type de la fonction et on crée un nouveau paramètre passage par variable du même type que la fonction ;

En appliquant les deux étapes a et b, la ligne n° 07 du programme précédent devient :

Procedure fact(n:integer ; var m:integer) ;

Le nouveau paramètre ici est m.

c- On remplace l'instruction *<id_fonction>:= <id_variable_resultat> ;* par l'instruction *<id_nouveau_paramètre>:= <id_variable_resultat> ;*

En appliquant l'étape c, nous la ligne N°15 du programme précédent devient :

m := f ;

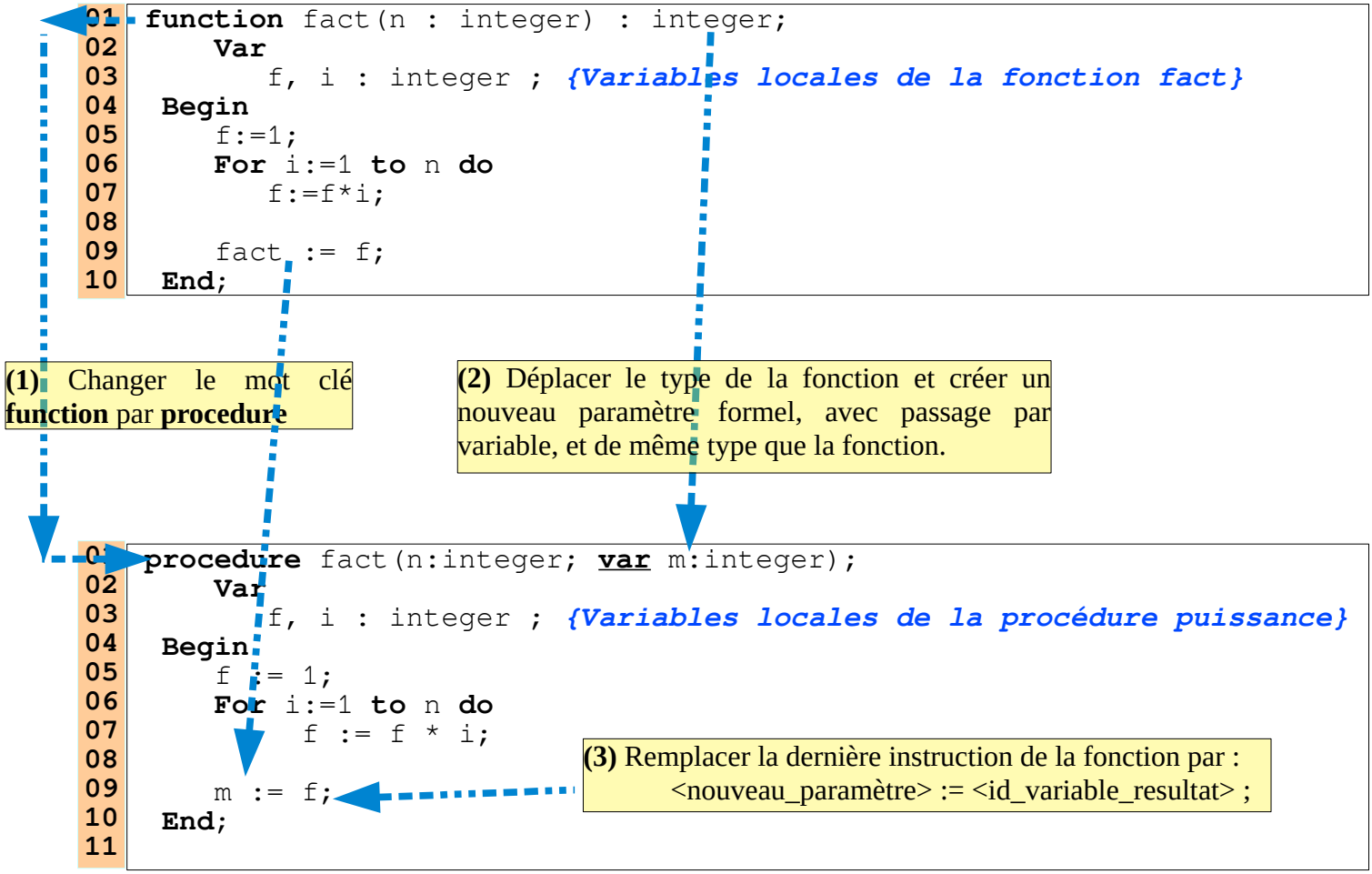
d- L'appel de chaque fonction, est remplacé par l'appel à la procédure, en ajouter le paramètre supplémentaire. On aura les nouvelles instructions suivantes :

fact (n, f1) ; fact (k, f2) ; fact (n-k, f3) ;

c- L'appel à la fonction est remplacé par la variable globale qui représente le paramètre effectif correspondant au passage par variable.

```
01 Program CombinaisonNK;
02 Uses wincrt ;
03 var
04     n, k, c : integer; {Variables Globales du programme}
05
06 Procedure fact(n:integer ; var m:integer) ;
07     Var
08        f, i : integer ; {Variables locales de la fonction fact}
09  Begin
10     F:=1;
11     For i:=1 to n do
12        f:=f*i;
13
14     m:=f;
15  End;
16
17 BEGIN {Début du Programme Principal}
18          Writeln('Donnez la valeur de n et k :');
19     Read(n, k);
20
21     fact(n,f1);
22     fact(k, f2);
23     fact(n-k, f3);
24
25     c := f1 div ( f2 * f3 );
26
27     Writeln('Le nombre de combinaisons de k à partir de n = ', c);
28 END. {Fin du Programme Principal}
29
```

Les étapes précédentes, pour transformer la fonction fact à une procédure (dans la partie déclaration) sont résumées sans la figure ci-dessous :



Nous pouvons transformer la fonction *fact* à une procédure sans ajouter un nouveau paramètre m. Et ceci en déplaçant la variable locale *f* (la variable résultat) comme paramètre avec passage par variable et en supprimer la dernière instruction de la fonction *fact:=f;* Comme indiquer sur la figure dans la page suivante :

```
01 function fact(n : integer) : integer;
02     Var
03        f, i : integer ; {Variables locales de la fonction fact}
04  Begin
05     f:=1;
06     For i:=1 to n do
07        f:=f*i;
08
09     fact := f;
10  End;
```

(1) Changer le mot clé **function** par **procedure**

(2) Déplacer le type de la fonction et la variable local résultat (la variable f) comme paramètre formel, avec passage par variable (logiquement le type de f est le même type que la fonction).

```
01 procedure fact(n:integer; var f:integer);
02     Var
03        i : integer ; {Variable locale de la procédure puissance}
04  Begin
05     f := 1;
06     For i:=1 to n do
07        f := f * i;
08
09  End;
```

(3) Supprimer la dernière instruction de la fonction :
fact := f ;

Le programme deviens comme suit (Les appels ne changent pas).

```
01 Program CombinaisonNK;
02 Uses wincrt ;
03 var
04     n, k, c : integer; {Variables Globales du programme}
05
06 Procedure fact(n:integer ; var f:integer) ;
07     Var
08        i : integer ; {Variables locales de la fonction fact}
09  Begin
10     f:= 1;
11     For i:=1 to n do
12        f:=f*i;
13  End;
14
15 BEGIN {Début du Programme Principal}
16          Writeln('Donnez la valeur de n et k :');
17     Read(n, k);
18
19     fact(n,f1);
20     fact(k, f2);
21     fact(n-k, f3);
22
23     c := f1 div ( f2 * f3 );
24
25     Writeln('Le nombre de combinaisons de k à partir de n = ', c);
26 END. {Fin du Programme Principal}
```

Dans le déroulement ci-dessous, nous utilisons le deuxième programme avec la procédure fact (avec paramètre *f* et non *m*).

3- Le déroulement du nouveau programme pour n=5 et k=4

Instructions	*Variables*								
	Programme Principal						*La procédure fact*		
	n	*k*	*fn*	*fk*	*fnk*	*c*	*n*	*i*	*f* (var)
Read(n, k)	**5**	**3**	**0**	**0**	**0**	**0**			
fact(n, fn) ;(l'appel à fact avec n=5 et fn = 0)	**5**	``	**0**						
=> *La transmission des paramètres → n=5* *f:=1* *for i=1 f:=f*i → f=01*1 = 1* *for i=2 f:=f*i → f=01*2 = 2* *for i=3 f:=f*i → f=02*3 = 6* *for i=4 f:=f*i → f=06*4 = 24* *for i=5 f:=f*i → f=24*5 = 120*							**5**	/ 1 2 3 4 5	0 1 1 2 6 24 120
Retour de résultat f vers fn	``	``	**120**						
fact(k, fk) ; (l'appel à fact avec k=3 et fk=0)	**5**	**4**	**120**	**0**	**0**				
=> *La transmission des paramètres → n=3* *f:=1* *for i=1 f:=f*i → f=1*1 = 1* *for i=2 f:=f*i → f=1*2 = 2* *for i=3 f:=f*i → f=2*3 = 6* *for i=4 f:=f*i → f=6*4 = 24*							**4**	/ 1 2 3 4	0 1 1 2 6 24
Retour de résultat f vers fk			**120**	**24**	**0**				
fact(n-k, fnk) ;(l'appel à fact avec n-k=1 et fnk = 0)	``	``	``	``	**0**				
=> *La transmission des paramètres → n=2* *f:=1* *for i=1 f:=f*i → f=1*1 = 1*							**1**	/ 1	0 1 1
Retour de résultat f vers fnk			**120**	**24**	**1**				
*C:= 120 div (24 * 1) → C = 120 div 24 = 5* *Writeln(c);*	``	``	``	``	``	**5** **5**			

Le programme affichera :

```
Le nombre de combinaisons de k à partir de n = 10
```

Bon Courage & Travaillez Bien.

_*_*_*_*_*_*_*_*_*_*_*_*_*_*_

EXERCICES SUPPLÉMENTAIRES

Exercices Supplémentaire : Procédure & Fonctions

Exercice 01 : Transformer une procédure à une fonction

Soit la déclaration d'une procédure puissance :

```
01 Procedure puissance(x:real; n:integer ; var p:real);
02     Var
03         i : integer ; {Variables locales de la procédure puissance}
04  Begin
05     P:=1;
06     For i:=1 to n do
07         p:=p*x;
08  End;
```

Et soient les appels suivants :

```
puissance(2.5, 3, z);
puissance(x+y, n div 2, f);
puissance(z - x / 3, n + m, a);
```

Tel-que : *x, y, z, f* et *a* sont des variables réels et *n* et *m* sont des variables entières.

- Transformer cette procédure puissance à fonction ? (Transformer la déclaration et l'appel).

Solution

Lorsque nous transformons une procédure à une fonction (et vice-versa : une fonction à une procédure), il faut réalisera la transformation :

- Au niveau de la déclaration
- Au niveau des appels à la fonction transformée (la nouvelle procédure).

- ***Au niveau de la déclaration***

nous suivons les étapes illustrées dans le schéma suivant :

```
01  Procedure puissance(x:real; n:integer ; var p:real);
02      Var
03          i : integer ; {Variables locales de la procédure puissance}
04   Begin
05      P:=1;
06      For i:=1 to n do
07          p:=p*x;
08   End;
```

(1) Changer le mot clé procedure par function

(2) Déplacer le paramètre avec passage par variable vers les variables locales de la fonction.

(3) Mettre le type de ce paramètre comme type de la fonction.

```
01  function puissance(x:real; n:integer) : real;
02      Var
03           i : integer ; {Variables locales de la fonction puissance}
04           P:real;
05   Begin
06      P:=1;
07      For i:=1 to n do
08                p:=p*x;
09
10      puissance := p;
11   End;
```

(4) Ajouter, à la fin de la fonction, l'instruction suivante : <id_fonction> := <id_variable_resultat> ;

- ***Au niveau des Appels***

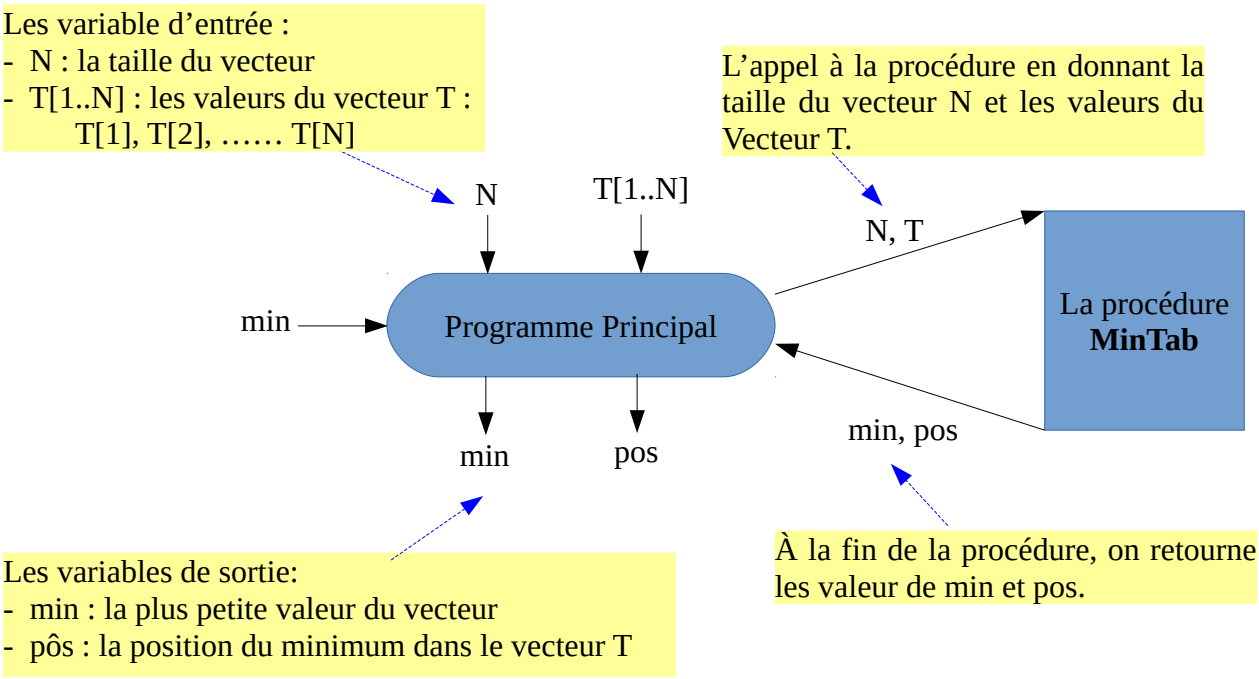
Appel à la Procédure	*Appel à la Fonction*
`puissance(2.5, 3, z);`	`Z := puissance(2.5, 3);`
`puissance(x+y, n div 2, f);`	`Z := puissance(x+y, n div 2);`
`puissance(z-x/ 3, n+m, a);`	`a := puissance(z-x/ 3, n+m);`

Exercice 2 : Minimum et sa position dans un vecteur

- Écrire un programme qui lit un **tableau** T de N réels, fait appel à une **procédure** qui détermine le plus petit élément du tableau ainsi que sa position (son rang dans le tableau).
- Écrire cette procédure et l'insérer dans le programme.
- Afficher les résultats dans le programme principal.

Solution

Le programme à réaliser, ainsi que la procédure, peuvent être schématisés comme suit :



D'après le schéma ci-dessus, nous avons :

- Deux variables d'entrées : *N* (la taille du vecteur) et *T* les valeurs des composantes
- Deux variables de sortie : *min* (la valeur du minimum) et *pos* (son rang) dans *T*.
- La procédure *MinTab* contient 4 paramètres :
- Deux paramètres d'entrée : un entier et un Tableau.
- Deux paramètres de sorties : un nombre réel et un entier.

Pour utiliser les tableaux (vecteurs ou matrices) comme paramètre dans un sous-programme (procédure ou fonction), il faut tout d'abord créer un nouveau type, dans la partie déclaration (avant les variables et sous programme) du programme principal, comme suit :

```
type VECTEUR = Array[1..100] of real;
type MATRICE = Array[1..10, 1..10] of real;
```

La procédure MinTab

Elle peut être définie comme suit :

```
01 Procedure MinTab(m:integer; V:VECTEUR ; var min:real; var pos:integer);
02     Var {Variables locales de la procédure MinTab}
03         i : integer ;
04  Begin
05     min := V[1];
06     pos := 1;
07     for i:=1 to m do
08          if V[i]<min then
09          begin
10              min := V[i];
11              pos := i;
12          end;
13  End;
```

Le programme complet

Le programme, avec le sous -programme ***MinTab***, sera écrit comme suit :

```
01 Program Exo3;
02 Uses wincrt ;
03
04  {Pour utiliser les vecteurs (ou les matrices) il faut déclarer un
05 nouveau type : ici, nous avons déclarer type VECTEUR}
06 Type VECTEUR = Array[1..100] of real;
07
08 Var {Variables Globales du programme}
09     T : VECTEUR;
10     n,i, pos : integer;
11     min : real;
12
13 Procedure MinTab(m:integer; V:VECTEUR; var min:real; var pos:integer);
14     Var {Variables locales de la procédure MinTab}
15          i : integer ;
16  Begin
17     min:=V[1];
18     pos:=1;
19     for i:=1 to m do
20               if V[i]<min then
21                begin
22                    min := V[i];
23                    pos := i;
24                end;
25  End; {La fin de la procédure MinTab}
26
27 BEGIN {Début du Programme Principal}
28     Write('Donnez la taille n :');
29     Read(n);
30     for i:=1 to n do
31               read(T[i]);
32
33     MinTab(n, T, min, pos);
34
35     Writeln('Le minimum = ', min:2:2, '  et sa position = ', pos);
36 END. {Fin du Programme Principal}
```

Les paramètres formels de la procédure **MinTab**

L'appel à la procédure **MinTab** avec les paramètres effectifs : **n, T, min et pos**
Les paramètre formels sont : *m, V, min, Pos*

À l'appel :
- La valeur de **n** est transmise à *m*
- Les valeurs de des composantes du vecteur **T** sont transmis aux composantes de **V**
- la valeur **min** est transmise à *min*.
- la valeur **pos** est transmise à *pos*.

À la fin de la procédure :
- La valeur *min* est retournée à **min**.
- La valeur *pos* est retournée à **pos**.

Remarque : paramètres formes & paramètres effectifs

Les paramètres effectifs d'un sous programme peuvent avoir le même nom (identificateur) que les paramètres formels, cependant, aucun lien entre ces identificateurs (noms). Les conditions qu'il faut vérifier lors de l'appel à un sous programme sont :

- le nombre de paramètres effectifs doit être égale au nombre de paramètres formels ;
- le type de chaque paramètre effectifs est le même que le paramètre formel correspondant.
- dans le passage par variable, nous devons mettre une variable comme paramètre effectif.

Pour consulter le programme Pascal et l'exécuter en ligne, voir :
https://onlinegdb.com/PvVGEVM7WQ

Exercice 3 : Développement Limité de l'Exponentiel.

Soit la somme suivante :

$$e = 1 + 1/1! + 1/2! + 1/3! + \ldots. + 1/n!$$

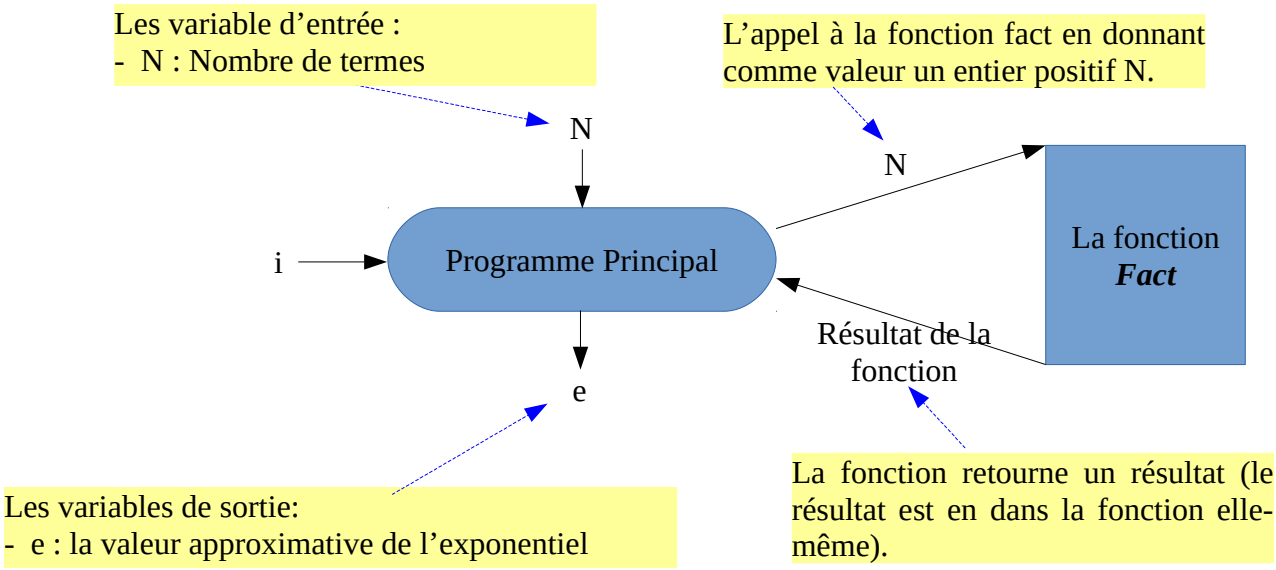
Écrire un programme PASCAL, qui fait appel à une fonction *FACT* pour calculer le factoriel d'un entier n, pour calcule la somme e. Afficher le résultat dans le programme principal.

(Pour la fonction *FACT* voir l'exercice N°02 page 06).

Solution

Le schéma du programme avec la fonction

Le programme à écrire, avec la fonction *fact*, peut être schématisé comme suit :



Analyse

La somme : e = 1 + 1/1! + 1/2! + 1/3! + …. + 1/n! Peut être écrite comme suit :

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots\ldots\ldots \frac{1}{n!} = 1 + \sum_{i=1}^{n} \frac{1}{i!}$$

En pseudo-algorithmique, on écrit :

```
e ← 1;
Pour i←1 à n faire
     e ← e / i!;
Fin-pour;
```

En algorithmique, et aussi en langage pascal, on ne peut écrire *i!* : on la remplace par l'appel à la fonction *fact* : *fact(i)*.

ça devient :

```
e ← 1;
Pour i←1 à n faire
     e ← e / fact(i);
Fin-pour;
```

Programme Pascal

```
01 Program Estimation_Exponentiel;
02 Uses wincrt ;
03 var
04     {Variables Globales du programme}
05     n, i : longint; {longint entre -2147483648 et 2147483647}
06     e : real;
07
08 Function fact(n:longint) : longint ;
09     Var
10        i : integer; {Variables locales de la fonction fact}
11        f : longint;
12  Begin
13     F:=1;
14     For i:=1 to n do
15        f:=f*i;
16
17     fact := f;
18  End;
19
20 BEGIN {Début du Programme Principal}
21     {Les entrées}
22     Write('Donnez la valeur de n  :');
23     Read(n);
24
25     {Traitement}
26     e := 1;
27     for i:=1 to n do
28        e := e + 1 / fact(i);
29
30     {Les sorties}
31     Writeln('Exponentiel e = ', e:0:4);
32 END. {Fin du Programme Principal}
```

Consulter le lien (Programme Pascal) : https://onlinegdb.com/5rX0nZR7o

Lien sur le développement des fonctions usuelles :

https://membres-ljk.imag.fr/Bernard.Ycart/mel/dl/node6.html

BON COURAGE & TRAVAILLEZ BIEN