Université Batna2
Département Informatique
Année universitaire 2021-2022

Algorithmique et structure de données 2

Plan du cours :

Chapitre 1 : Les sous-programmes : Fonctions et Procédures

Chapitre 2 : Les fichiers

Chapitre 3 : Les listes chaînées

Madame Malika Bachir - première année, deuxième semestre 2021-2022. Batna

1 Introduction

Dans la programmation (l'algorithmique), dès qu'on commence à écrire des programmes importants, il devient difficile d'avoir une vision globale sur son fonctionnement et de traquer les erreurs ; ainsi dans certains programmes on trouve une répétition de résoudre un problème donné dans plusieurs reprises.

Par exemple, dans le même algorithme il nous a demandé de calculer :

1. La factorielle du maximum d'un tableau
2. La factorielle du maximum d'une matrice de n lignes et m colonnes
3. La factorielle du maximum d'une matrice de m lignes et n colonnes
4. n^m
5. m^n

Que faire pour simplifier la réalisation, la correction, la maintenance, … de ce programme ? $\Rightarrow$ ***Décomposer*** le problème en sous problèmes et trouver une solution à chacun puis ***regrouper*** le tout dans un seul programme (algorithme). En Algorithmique (programmation), chaque solution partielle donne lieu à un sous-algorithme (sous-programme) qui fera partie d'un algorithme complet pour pouvoir être exécuté.

2 Définitions

Un sous-algorithme est un bloc faisant partie d'un algorithme. Il est **déclaré** dans la **partie entête** (avant le début de l'algorithme) puis **appelé** dans le corps de l'algorithme.

Comme le sous-algorithme est un bloc à part entière, il possède alors un en-tête, une série de traitements, et une gestion des résultats tout comme l'algorithme qui le contient.

Un sous-algorithme est déclaré de manière générale c.-à-d. qu'il peut être appelé plusieurs fois avec différentes valeurs grâce à des arguments. Ces derniers, bien qu'ils soient facultatifs, sont dits paramètres et sont clairement déclarés, au besoin, dans l'entête du sous-algorithme.

Un paramètre est une valeur du bloc principal dont le sous-algorithme a besoin pour exécuter avec des données réelles l'enchaînement d'actions qu'il est chargé d'effectuer. On distingue deux types de paramètres:

- **Les paramètres formels** sont la définition du **nombre et du type** de valeurs que devra recevoir le sous-algorithme pour se mettre en route avec succès. On déclare les paramètres formels pendant la déclaration du sous-algorithme.
- **Les paramètres effectifs** sont des **valeurs réelles** (constantes ou variables) reçues par le sous-algorithme au cours de l'exécution du bloc principal. On les définit indépendamment à chaque appel du sous-algorithme dans l'algorithme principal.

L'exécution d'un sous-algorithme (procédure ou fonction) se fait par une **instruction d'appel**. L'application de cette instruction génère un saut vers le sous-algorithme appelé. La terminaison de ce sous-algorithme redémarre la suite d'instruction interrompue par l'appel.

3 Les variables locales et les variables globales

Un sous-algorithme utilise les variables déclarées dans l'algorithme (appelées variables globales). Il peut aussi avoir ses propres variables (dites locales) déclarées dans l'espace qui lui est réservé ; mais qui ne peuvent être utilisées que dans ce sous-algorithme et nulle part ailleurs car sa portée (visibilité) est limitée au bloc qui la contient. L'espace de ces variables locales n'est réservé que lorsque le sous-algorithme est appelé et est libéré dès la fin de l'exécution.

6) Types de sous-algorithme

Un sous-algorithme peut se présenter sous forme de fonction ou de procédure.

Une fonction est un sous-algorithme qui, à partir de donnée(s), **calcul et rend** à l'algorithme Un et Un seul résultat alors qu'en général, une procédure **affiche** le(s) résultat(s) demandé(s).

a) Fonction

Une fonction est un bloc d'instructions qui retourne obligatoirement une et une seule valeur résultat à l'algorithme appelant. Une fonction n'affiche jamais la réponse à l'écran car elle la renvoie simplement à l'algorithme appelant.

➢ **Déclaration d'une fonction :**

```
Fonction Nom_Fonction (Nom_Paramètre : Type_praramètre;……) : type_Fonction
                                  Facultatifs
Déclaration
Nom_variable : Type_variable                                  { Variables locales }
…
Début
    …
    Instructions ; Corps de la procédure
    …
    Nom_Fonction ← Résultat  / ou retourner (Résultat)        {Obligatoire}
FinFonctio
```

Etant donné qu'une fonction a pour but principal de renvoyer une valeur, il est donc nécessaire de préciser **le type de la fonction** qui est en réalité **le type de cette valeur** (le résultat).

➢ **Appel d'une fonction :**

Un appel de fonction est une expression d'affectation de manière à ce que le résultat soit récupéré dans une variable globale : **Nom_variable-globale ← Nom_Fonction (paramètres) ;**

Exemple : Voici un algorithme utilisant une fonction qui calcule une somme de 100 nombres.

```
Algorithme essai
Variable
I, Som : entier       ( variable globale )
Fonction Somme : entier
Variable
S : entier            ( Variables locales )
Debut /*Début de la fonction*/
    S ← 0
    Pour I de 1 à 100 Faire
        S ← S + I
    FinPour
    Somme ← S  (ou retourner S)
Fin /*Fin de la Fonction */
Debut /*Début de l'algorithme*/
    Som ← Somme     (l'appel du fonction)
    Ecrire ('La somme des ', N, 'premiers nombres est', Som)
Fin /*Fin de l'algorithme*/
```

Note : Une fonction peut appeler d'autres sous-algorithmes à condition qu'ils soient définis avant elle ou qu'ils soient déclarés dans son entête.

b) Procédure

Une procédure est un bloc d'instructions nommé et déclaré dans l'entête de l'algorithme et appelé dans son corps à chaque fois que le programmeur en a besoin. La procédure ressemble parfaitement à la fonction sauf que cette dernière peut renvoyer zéro ou plusieurs résultats.

- **Déclaration d'une procédure :**

```
Procédure Nom_Procédure (Nom_Paramètre : Type_praramètre;……)
                                        Facultatit
Déclaration
Nom_variable : Type_variable                {Variables locales }
…
Début
    …
    Instructions ; Corps de la procédure
    …
Fin
```

- **Appel d'une procédure :**

L'appel d'une procédure peut être effectué en spécifiant, au moment souhaité, son nom et éventuellement ses paramètres ; cela déclenche l'exécution des instructions de la procédure.

Exemple : L'algorithme précédent, qui calcule une somme de N nombres, peut utiliser une procédure au lieu d'une fonction.

```
Algorithme essai
Variable
I, S : entier          (variable globale)
Procedure Somme
Debut /*Début de la Procédure*/
    S ← 0
    Pour I de 1 à 100 Faire
        S ← S + i
    FinPour
    Ecrire ('La somme des 100 premiers nombres est', S)
Fin /*Fin de la Procédure*/
Debut /*Début de l'algorithme*/
    Somme      (Appel de la procédure)
Fin /*Fin de l'algorithme*/
```

7) Le passage des paramètres

Il existe deux types de passage de paramètres : par **valeur** et par **variable** (dite aussi par référence ou encore par adresse).

a) Passage de paramètres par valeur

- C'est le mode de transmission par défaut, il y a copie de la valeur des paramètres effectifs dans les variables locales issues des paramètres formels de la procédure ou de la fonction appelée.
- Dans ce mode, le contenu des paramètres effectifs ne peut pas être modifié par les instructions de la fonction ou de la procédure ; car nous ne travaillons pas directement avec la variable, mais sur une copie.

- À la fin de l'exécution du sous-algorithme la variable conservera sa valeur initiale. Les paramètres dans ce cas sont utilisés comme données.

Syntaxe :

Procedure nom_procédure (param1 :type1 ; param2, param3 :type2)

Fonction <nom_fonction> (param1 :type1 ; param2 :type2) : Type_fonction

Exemple : Soit l'algorithme suivant.

```
Algorithme pas-val
variable
M : entier
Procedure P1 (nombre : entier)
Début
    Si nombre > 0 Alors
        nombre ← nombre*2
    FinSi
    Ecrire (nombre)
Fin
Début
    Lire (M)
    P1 (M)
    Ecrire (M)
Fin
```

En exécutant cet algorithme pour M = 5, nous obtenons la trace d'exécution suivante :

Algorithme pas-val

M	Sortie
5	**5**

Procédure P1

nombre	Sortie
5 10	10

- Au début d'exécution de l'algorithme, la variable M reçoit la valeur 5
- Ensuite, l'algorithme appelle la procédure P1 par le paramètre effectif M
- L'exécution de la procédure P1 affiche comme résultat la valeur 10
- Au retour à l'algorithme (au niveau de l'appel) il ne reste que d'afficher la variable globale M avec sa valeur initiale 5. Donc, l'algorithme affiche sur écran en ordre : 10 puis 5, càd M n'a pas changé de valeur (M n'a pas reçu le résultat).

- **Nous concluons que le passage par valeur ne change pas le contenu du paramètre effectif.**

b) Passage de paramètres par variable

Ici, il s'agit non plus d'utiliser simplement la valeur de la variable, mais également son emplacement dans la mémoire (d'où l'expression « par adresse »). En fait, le paramètre formel se substitue au paramètre effectif durant le temps d'exécution du sous-programme et à la sortie il lui transmet sa nouvelle valeur. D'une autre façon, il s'agit de transmettre **l'adresse de la variable** du paramètre effectif et non sa valeur, alors toute opération se fait directement dans cette variable.

Un tel passage de paramètre se fait par l'utilisation du mot-clé **Var** (ou référence).

Syntaxe :

Procedure nom_procédure (**Var** param1 :type1 ; param2, param3 :type2)

Fonction <nom_fonction> (**Var** param1 : type1 ; param2 :type2) : Type_fonction

Note : Les paramètres passés par valeur et par adresse peuvent cohabiter à l'intérieur d'un même sous-algorithme. Il suffit de partager les deux types de passage par un (**;**).

Syntaxe :

Procedure nom_procédure (**Var** param1 :type1 ; param2, param3 :type2)

Dans ce cas param1 est passé par référence alors que les deux autres sont passés par valeur

Fonction <nom_fonction> (param1 :type1 ; **Var** param2 :type2) : Type_fonction

Dans ce cas param1 est passé par valeur alors que le deuxième est passé par référence

Exemple :

Prenons l'algorithme précédent en modifiant le type de passage de paramètre en passage par adresse

```
Algorithme pas-val

variable
M : entier
procedure P1 (Var nombre : entier)
Début
    Si nombre > 0 Alors
        nombre ← nombre*2
    FinSi
    Ecrire (nombre)
Fin
Début
    Lire (M)
    P1 (M)
    Ecrire (M)
Fin.
```

Exécutons cet algorithme toujours pour la valeur (5), la trace d'exécution sera donc comme suit :

Algorithme pas-val

@ : 143	
M	Sortie
5 10	**10**

Procédure P1

nombre	Sortie
@ : 143	10

- Au début d'exécution de l'algorithme, la variable M reçoit la valeur 5
- Ensuite, l'algorithme appelle la procédure P1 par le paramètre effectif M (passage par adresse)
- L'exécution de la procédure P1 affiche comme résultat la valeur 10
- Au retour à l'algorithme (au niveau de l'appel) il ne reste que d'afficher la variable globale M qui est dans ce cas 10. Donc, l'algorithme affiche sur écran en ordre : 10 puis 10, càd M a changé sa valeur initial (M a reçu le résultat).

➢ **Nous concluons que le passage par adresse change le contenu du paramètre effectif.**

8) La récursivité

a) Définition

Un algorithme (une fonction, une procédure) est dit récursif si sa définition (son code) contient un appel à lui-même.

Un algorithme qui n'est pas récursif est dit itératif.

b) Utilisations usuelles

Utilisations variées (liste non exhaustive) :

- Calcul de suite récursive (numérique, graphique. . .)
- Calcul de type « diviser pour régner » : recherche, tri, . . .
- Calcul sur des structures de données inductives (listes, arbres, . . .)
- Dans tous les cas, une version itérative est possible.

c) Quelques exemples classiques

1) Factorielle : $n! = n.(n-1)!$

```
Fonction fact(n) : entier
Début
    Si n=0 alors
        fact ← 1   ( ou Retourner 1)
    Sinon
        fact ← n*fact(n-1)  ( ou Retourner n*fact(n-1)  )   {Appel récursif}
    Fsi
FFonction
```

2) Fibonacci : $Fibo(n) = Fibo(n-1) + Fibo(n-2)$

```
Fonction fibo(n) : entier
Début
   Si n=0 alors
      fibo ← 1 (ou Retourner 1 )
   Sinon
      Si n=1 alors
         fibo ← 1 (Retourner 1)
      Sinon
         fibo ← fibo(n-1)+fibo(n-2)  (ou Retourner fibo(n-1)+fibo(n-2) ) {Appel récursif}
      Fsi
   Fsi
FFonction
```

3) Que calcule somme(5,0) ?

```
Fonction somme (n,r) :entier
Début
   Si n = 1 alors
      Somme ← 1 ( ou Retourner r + 1)
   Sinon
      Somme ← somme (n -1 , r + n ) ( ou Retourner somme (n -1 , r + n ))   {Appel récursif}
   Fsi
FFonction
```

9) Fonctions en C

a) Définition

type_de_retour nom_fonction(liste-params)

{

liste-declarations (optionnelle)

liste_instructions

return (**du type type_de_retour**)

}

La liste d'instructions comprend au moins une instruction return (du type type_de_retour).

b) Appel

nom_fonction(liste-expressions)

c) Exemple

Définition d'une fonction

```
int max ( int a , int b )
{
        int m;
        i f (a>b)
                m=a ;
        else
                 m=b ;
        return (m) ;
}
```

Appel

T = max(3 ,45);

En utilisant cette fonction pour afficher le maximum d'un tableau, nous écrivons alors le programme suivant :

```
#include<stdio.h>
int max ( int a , int b )
{
    int m;
    if (a>b)
      m=a ;
    else
       m=b ;
    return (m) ;
}
main()
{
    int i, M, T[100] ;
    for(i=0 ; i<10 ; i++)
      scanf("%d", &T[i]) ;
    M=T[0] ;
    for(i=1 ; i<10 ; i++)
      M=max(M , T[i]) ;  // appel du fonction
    printf("Le maximum du tableau est : %d", M) ;
}
```

10) Procédures en C

a) Définition

```
void nom_action(liste-params)
{
    liste-declarations (optionnelle)
    liste_instructions
}
```

b) Appel

```
nom_action(liste-expressions)
```

- **On ne récupère pas le résultat d'une procédure, il n'y en a PAS.**

c) Exemple

Reprenons le même exemple :

Définition de la procédure

```
void max(int a , int b)
{
        int m;
        i f (a>b)
                m=a ;
        else
                 m=b ;
        printf(''%d'', m) ;
}
```

Appel

```
max(3 ,45);
```

Pour utiliser cette procédure pour afficher le maximum d'un tableau, nous devons utiliser le passage des paramètres par adresse ; alors le programme s'écrit comme suit :

```
#include<stdio.h>
void max ( int a , int b , int *c)
{
int m;
if (a>b)
m=a ;
else
 m=b ;
*c=m;
}
main()
{
int i, M, T[10] ;
for(i=0 ; i<10 ; i++)
scanf("%d", &T[i]) ;
M=T[0] ;
for(i=1 ; i<10 ; i++)
max(M , T[i], &M) ;  // appel de la procédure
printf("Le maximum du tableau est : %d", M) ;
}
```

Pour le passage de paramètre par valeur, il n'y a pas de nouveau comme le montre l'exemple de définition et d'appel de procédure ; tandis qu'au passage de paramètre par adresse, le/les paramètres formelles qu'on

désir renvoyer le/les résultats sont précédés après le/leurs types par **(*)** et le/les paramètres effectifs qui reçoivent ce/ces résultats sont précédés par **(&).**

Exemple :

```
#include<stdio.h>
void fact(int y ,int *z)
{
   int t=1, j;
   for(j=1 ; j<=y ; j++)
      t=t*j ;
     *z=t ;
}
main()
{
  int b;
  fact(4,&b);
  printf("%d", b);
}
```

d) Exercices

Écrire les procédures suivantes :

- Impression du maximum de 3 entiers en paramètres
- Impression des 100 premiers termes de la suite suivante : u0 = 32, un = 3∗ un−1 + 19
- Impression des k premiers termes de la suite, avec k passé en paramètre.

11) Passage de paramètres par adresse dans les fonctions

Pour renvoyer plus d'un résultat dans les fonctions, l'un des résultats est retourné par return() ; les autres sont renvoyés en utilisant le passage par adresse tous comme les procédures.

Exemple : calculer la somme et le produit d'une suite allant de 1 à a, càd : (1+2+ ….+a) et (1*2* ….*a)

```
#include<stdio.h>
int fact(int x, int *y)
{  int r,r2,i;
  r=1 ;
  r2=0 ;
  for(i=1;i<=x;i++)
  {
     r=r*i;
     r2=r2+i;
  }
  *y=r2;
  return(r);  // se place en dernier
}
main()
{
  int a=4,b;
  a=fact(a , &b);  printf( "%d%d" ,  a , b ); // après exécution affiche deux résultats : 24 et 10
}
```

RQs :

- On déclare "valeur" un paramètre si on ne veut pas que l'appel modifie les arguments (paramètres effectifs)
- On déclare "référence" un paramètre si on veut que l'appel modifie les arguments
- Bien sûr, on peut mélanger paramètres par valeur et par référence.

Exercices Corrigés : Les fonctions et procédures

(UNIVERSITE CONSTANTINE 2 TRONC COMMUM - MI 2014/2015)

Exercice 1 :
Ecrire une fonction ou procédure qui calcule la partie entière d'un nombre positif.

```
Fonction entiere (x : reel) : entier ;
Declaration
Variable
        y : entier ;
Debut
        y← 0 ;
        Tantque y < x faire
                y ← y + 1
        FinTanque ;
        entiere ← y
Fin ;
```

Exercice 2 :
Ecrire une fonction ou procédure qui affiche le tableau de multiplication d'un entier positif x.

```
Procedure Multipl (x: entier) ;
Declaration
Variable
        a, b : entier ;
Debut
        b ← 0 ;
        Pour a ← 1 a 10 faire
          Debut
                b ← b + x
                Ecrire (x, 'X', a, '= ', b) ;
          Fin
        FinPour
Fin ;
```

Exercice 3 :
Ecrire une fonction ou procédure qui calcule le PGCD de deux entiers strictement positifs.

```
Fonction PGCD (a, b : entier) : entier ;
Debut
        Tantque (a * b ≠ 0) Faire
                Si a > b Alors
                    a ← a - b
                Sinon
                    b ← b - a ;
                FinSi
        FinTantque ;
        Si a = 0 alors
           PGCD ← b
        Sinon
           PGCD ← a
        FinSi
Fin ;
```

Exercice 4 :

Ecrire une fonction ou procédure qui permet de lire deux nombres, calculer la somme et le produit et affiche si ces derniers sont positifs ou négatifs.

```
Procedure calcul ;
Declaration
Variable
        a, b, som, prod : reel ;
Debut
        Lire (a, b) ;
        som ← a + b ;
        prod ← a * b ;
        Si som ≥ 0 Alors
                Ecrire ('la somme est positive')
        Sinon
                Ecrire ('la somme est négative')
        FinSi;
        Si prod ≥ 0 Alors
                Ecrire (' et le produit est positif')
        Sinon
                Ecrire (' et le produit est négatif')
        FinSi
Fin ;
```

Exercice 5 :

Ecrire une fonction ou procédure qui permet de lire une liste de nombres entiers dont la dernière valeur = -1 et affiche le nombre d'entiers pairs et leur pourcentage par rapport au nombre d'entiers donnés.

```
Procedure Nombres ;
Declaration
Variable
        x, cop, co: entier;
        pourcent: reel ;
Debut
        cop ← 0;
        co ← 0;
        Repeter
                Lire (x) ;
                co ←co + 1;
                Si x mod 2 = 0 alors
                    cop ← cop + 1;
                FinSi;
        Jusqu'à x = -1 ;
        pourcent ← cop * 100 / co;
        Ecrire ('Nombre de valeurs paires = ', cop, 'et leur pourcentage = ', pourcent)
Fin ;
```

Exercice 6 :

Ecrire une fonction ou procédure qui permet d'entrer deux valeurs M et N et d'afficher toutes les valeurs paires entre M et N si M < N.

```
Procedure calcul ;
Declaration
Variable
        M, N : entier ;
Debut
        Lire (M, N);
        Si M ≥ N Alors
                Ecrire ('Pas d"affichage')
        Sinon
                Tantque M < N Faire
                Debut
                        Si M mod 2 = 0 Alors
                                Ecrire (M)
                        FinSi ;
                        M ← M + 1
                Fin
                FinTantque
        FinSi
Fin ;
```

Exercice 7 :
Ecrire une fonction ou procédure qui affiche si un nombre est premier ou non

```
Procedure premier (a : entier) ;
Declaration
Variable
        b : booleen ;
        d : entier ;
Debut
        b ← vrai ;
        d ← 2 ;
        Tantque (d ≤ a/2) et (b = vrai) Faire
                Si a mod d = 0 Alors
                        b ← Faux
                Sinon
                        d ← d + 1
                FinSi ;
        FinTantque
        Si d = vrai Alors
                Ecrire (a, 'est premier')
        Sinon
                Ecrire (a, 'n"est pas premier')
        FinSi
Fin ;
```

Exercice 8 :
Ecrire une fonction ou procédure qui affiche tous les nombres pairs compris entre deux valeurs entières positives lue x et y

```
Procedure calcul ;
Declaration
Variable
        x, y, z : entier ;
Debut
        Lire (x, y);
        Si x > y Alors
          Debut
            z ← x ;
            x ← y ;
            y ← z ;
          Fin
        FinSi ;
        Tantque x ≤ y Faire
                Si x mod 2 = 0 Alors
                    Ecrire (x)
                FinSi ;
                x ← x + 1
        FinTantque
Fin ;
```

Exercice 9 :

Ecrire une fonction ou procédure qui permet d'entrer la date d'aujourd'hui puis demande le nom de la personne ; si ce nom = Ahmed il y a affichage de "Bienvenue Ahmed » puis lui demande sa date d'anniversaire et la compare à la date d'aujourd'hui si c'est la même il y a affichage de "Joyeux Anniversaire Ahmed" sinon il y a affichage "erreur de personne !"

```
Procedure Bienvenue ;
Declaration
Variable
        jj, mm, aa, ja, ma, aan : entier ;
        nom : chaine de caractère ;
Debut
        Ecrire (SVP donnez la date d'aujourd"hui') ;
        Lire (jj, mm, aa) ;
        Ecrire (SVP quel est votre nom ?') ;
        Lire (nom) ;
        Si nom = 'Ahmed' Alors
          Debut
                Ecrire ('Bienvenue Ahmed') ;
                Ecrire (quelle est la date de votre anniversaire ?') ;
                Lire (ja, ma, aan) ;
                Si (ja = jj) et (ma = mm) et (aan = aa) Alors
                    Ecrire ('Joyeux Anniversaire Ahmed')
                FinSi
          Fin
        Sinon
                Ecrire ('Erreur de personne')
        FinSi
Fin ;
```

Exercice 10 :
Ecrire une fonction ou procédure qui permet de résoudre une équation du second degré dans R.

```
Procedure Second-deg (a, b, c : entier) ;
Declaration
Variable
    delta, x1, x2; reel;
Debut
    Si a = 0 Alors
        Si b = 0 Alors
            Si c = 0 Alors
                Ecrire ('R est la solution')
            Sinon
                Ecrire ('Impossible')
            FinSi
        Sinon
            Debut
                x1 ← - c / b ;
                Ecrire (x1) ;
            Fin
        FinSi
    Sinon
        Debut
            delta ←b*b – 4*a*c ;
            Si delta < 0 Alors
                Ecrire ('Pas de solution dans R')
            Sinon
                Si delta = 0 Alors
                    Debut
                        x1 ← -b / 2*a ;
                        Ecrire ('Solution double ', x1) ;
                    Fin
                Sinon
                    Debut
                        x1 ← - b + √delta / 2*a ;
                        x2 ← - b - √delta / 2*a,
                        Ecrire (x1, x2) ;
                    Fin
                FinSi
            FinSi
        Fin
    FinSi
Fin ;
```

Exercice 11 :
Ecrire une fonction ou procédure qui permet de calculer la multiplication de deux nombres A et B entiers en utilisant l'addition.

```
Fonction multiple (A, B : entier) : entier ;
Declaration
Variable
  Res, Y : Entier ;
Debut
   Res ← 0 ;
   Si B < 0 Alors
      Y ← -B
   Sinon
    Y ← B
   FinSi
   Tantque Y > 0 Faire
     Debut
       Res ← Res + A
       Y ← Y – 1
     Fin
   FinTanque ;
   Si B < 0 Alors
     Res ← - Res
   FinSi ;
   multiple ← Res ;
Fin ;
```

Exercice 12 :
Ecrire une fonction ou procédure qui permet d'avoir un nombre entier positif et afficher son image miroir. Exemple le nombre est 3524, on doit afficher 4253.

```
Procedure Miroir (x : entier) ;
Declaration
Variable
        a, b : entier ;
Debut
        Tantque x ≠ 0 Faire
          Debut
                a ← x Mod 10 ;
                Ecrire (a) ;
                x ← x Div 10 ;
          Fin
        Fintantque
Fin ;
```

Exercice 13 :
Ecrire un algorithme (en utilisant fonction et/ou procédure) qui permet de calculer le cosinus de x € [0, Π/2] sachant que : Cos(W) = 1 – P(W, 2)/2! + P(W,4)/4! – P(W,6)/6! …

Pour réaliser cette fonction nous allons écrire deux fonctions générales Fact et P qui réalisent respectivement le calcul de la fonctionnelle d'un nombre entier et celle qui produit X^Y, X étant un réel et Y un entier.

```
Fonction Fact (N: entier) : entier ;
Declaration
Variable
  R, Y : Entier ;
Debut
  R ← 1 ;
  Tantque N > 1 Faire
    Debut
      R ← R * N;
      N ← N – 1;
    Fin
  FinTantque ;
Fact ← R
Fin ;
```

```
Fonction P (X : réel, Y : entier) : Réel ;
Declaration
Variable
  Z : Entier ;
  R : Réel ;
Debut
  Si X =0 Alors
    R ← 0
  Sinon
    Début
      R ← 1 ;
      Si Y < 0 Alors
        Z ← -1
      Sinon
        Z ← 1
      FinSi
      Tantque Y > 0 Faire
        Debut
          R ← R * X
          Y ← Y – 1
        Fin
      FinTantque
      Si Z < 0 Alors
        R ← 1 / R
      FinSi
    Fin
  P ← R
Fin ;
```

```
Fonction Cosinus (W : reel, Nb : entier) : Reel ;
Declaration
Variable
  F, Co, F1 : Entier ;
  Res, Y : Réel ;
Debut
  Res ← 1 ;
  F1 ← -1 ;
  Pour Co allant de 2 à Nb Faire
    Debut
      F ← Fact (Co);
      Y ← P (W, Co);
      Res ← Res + Y / F * F1;
      F1 ← - F1 ;
    Fin
  FinPour ;
  Cosinus ←Res
Fin ;
```