Ecole **I**nternationale des **S**ciences du **T**raitement de l'**I**nformation
Classe : 1ière année

Corrigé de la série d'exercices N° 5

RECURSIVITE

---

1. Calcul de la factorielle :

Calcule de la factorielle d'un nombre N donné par l'utilisateur

a. version itérative

```
with Ada.Text_Io;              use Ada.Text_Io;
with Ada.Integer_Text_Io; use Ada.Integer_Text_Io;
procedure Facto_I is
    N : Natural; -- valeur dont on veut calculer la factorielle
    function Factorielle ( N : Natural ) return Natural is
        Tempo : Natural := 1 ; -- variable de travail
    begin -- Factorielle
        -- Realiser toutes les multiplications
         for I in 2 .. N loop
            Tempo := Tempo * I ;
         end loop;
        return Tempo ;
     end Factorielle;
begin – Facto_I
   Put_Line ( "CALCUL DE N FACTORIELLE " );
   Put_Line ( " INTRODUISEZ N >= 0, RAISONNABLE " );
   Get ( N ); Skip_line;
   Put ( "N! = " );
   Put ( Factorielle ( N ) );
   New_Line;
   -- Pour terminer
    Put_line ( "Pressez <Enter> pour terminer" ); Skip_Line;
end Facto_I;
```

b. version recursif

```
with Ada.Text_Io; use Ada.Text_Io;
with Ada.Integer_Text_Io; use Ada.Integer_Text_Io;
procedure Facto_R is
     N : Natural; -- valeur dont on veut calculer la factorielle
     function Factorielle ( N : Natural ) return Natural is
     begin -- FACTORIELLE
          if N = 0 then
```

```
            return 1; -- cas limite simple
        else
            return N * Factorielle ( N -1 ); -- appel recursif
        end if;
    end Factorielle;

begin -- Facto_R
    Put_Line ( "CALCUL DE N FACTORIELLE " );
    Put_Line ( " INTRODUISEZ N >= 0, RAISONNABLE " );
     Get ( N ); Skip_Line;
     Put ( "N! = " );
     Put ( Factorielle ( N ) );
      New_Line;
     -- Pour terminer
      Put_line ( "Pressez <Enter> pour terminer" ); Skip_Line;
end Facto_R;
```

2. Calcul de C_n^p

Une première solution simple :

```
with Ada.Text_Io, Ada.Integer_Text_Io;
use  Ada.Text_Io, Ada.Integer_Text_Io;

procedure Combinaison is

   function Calcul_Combi(N, P : integer) return Integer is
   begin
      if P = 0 then
         return 1;
      elsif P = N then
         return 1;
      else
         return Calcul_Combi(N-1, P-1) + Calcul_Combi(N-1, P);
      end if;
   end Calcul_Combi;

   N, P : Integer;
begin
   loop
      Put("valeur de N :");
      Get(N);
      exit when N = 0;
      Put("valeur de P (inferieur ou egal a N): ");
      Get(P);
      Put("le nombre de combinaison de N elements P a P est ");
      Put(Calcul_Combi(N, P));
      New_Line;
   end loop;
end Combinaison;
```

Faire remarquer le double appel récursif.

Remarque :

Que se passe-t-il si on entre P plus grand que N ? ➔ segmentation fault (erreur d'exécution)

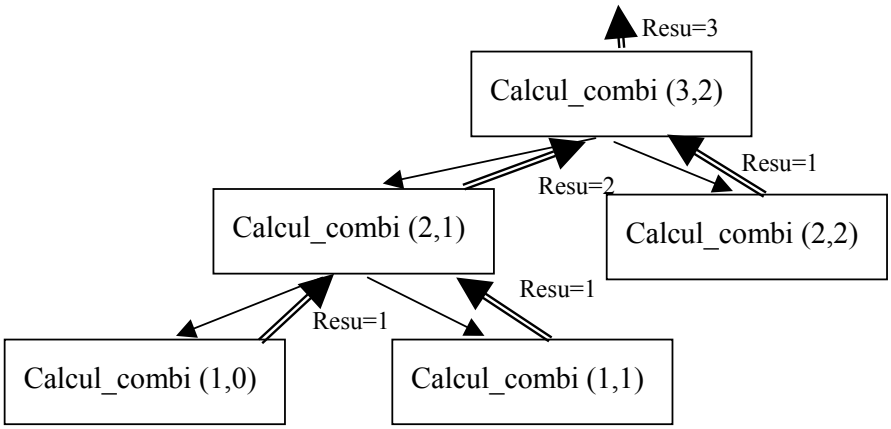
On peut aussi déclencher une exception dans le sous programme récursif lorsque P est plus grand que N et ensuite traiter cette exception dans le programme appelant.

```
with Ada.Text_Io, Ada.Integer_Text_Io;
use  Ada.Text_Io, Ada.Integer_Text_Io;

  Erreur_Combinaison : exception;

procedure Combinaison is

   function Calcul_Combi(N, P : integer) return Integer is
   begin
      if P <= N then
         if P = 0 then
            return 1;
         elsif P = N then
            return 1;
         else
            return Calcul_Combi(N-1, P-1) + Calcul_Combi(N-1, P);
         end if;
      else
         raise Erreur_Combinaison;
      end if;
   end Calcul_Combi;

begin
   loop
      Put("valeur de N :");
      Get(N);
      exit when N = 0;
      Put("valeur de P (inferieur ou egal a N): ");
      Get(P);
      begin
          Put("le nombre de combinaison de N elements P a P est ");
          Put(Calcul_Combi(N, P));
          New_Line;
      exception
          when Erreur_Combinaison =>
                       New_Line;
                       Put_Line(" ATTENTION P <= a N");
      end;
   end loop;
end Combinaison;
```

Arbre d'appel

On a N = 3 et P = 2

Arbre d'appel : les nœuds représentent les appels à la fonctions récursives; les arcs représentent la manière dont est traité un appel



3 Nombre de Fibonacci

a. version récursive

```
function Fibonacci ( N : Natural ) return Natural is
begin -- Fibonacci
        case N is
                when 0 | 1 => return N;        -- Cas de F0 et F1
                when others =>                 -- Appels recursifs
                        return Fibonacci ( N - 1 ) + Fibonacci ( N - 2 );
        end case;
end Fibonacci;
```

b. version itérative

```
function Fibonacci ( N : Natural ) return Natural is
        Avant_Dernier : Natural := 0; -- Avant-dernier nombre calcule
        Dernier : Natural := 1;       -- Dernier nombre calcule
        Courant : Natural;            -- Nombre a calculer
begin -- Fibonacci
        case N is
                when 0 | 1          => return N; -- Cas de F0 et F1
                when others    =>                -- Calculer le nombre demande
                        for I in 2 .. N loop     -- Fn = Fn-1 + Fn-2
                                Courant := Dernier + Avant_Dernier;
                                Avant_Dernier := Dernier;
                                Dernier := Courant;
```

```
            end loop;
            return Courant;
    end case;
end Fibonacci;
```

4 Palindrome

On va rechercher si un mot donné est ou non un palindrome. Dans un premier temps on ne considère que des mots (pas d'espace, de signe de ponctuation, etc).

```
function palindrome(Mot : in string) return boolean is
begin
   if Mot'length <= 1 then
      -- condition d'arret : 0 ou 1 seule lettre
      return true ;
   elsif Mot(Mot'first) /= Mot(Mot'last) then
      -- condition d'arret : différence des lettres extrêmes
      return false ;
   else -- les 2 lettres extrêmes sont égales
        --appel récursif en enlevant les lettres extrêmes
      return palindrome(Mot(Mot'first+1..Mot'last-1)) ;
   end if ;
end palindrome ;
```

Exemple : avec le mot laval

```
1/ Palindrome(laval)
   2/ Palindrome(ava)
      3/ Palindrome(v)
      3/ true
   2/ true
1/ true
```

avec le mot 123421

```
1/ Palindrome(123421)
   2/ Palindrome(2342)
      3/ Palindrome(34)
      3/ false
   2/ false
1/ false
```

5 Somme des éléments d'un tableau

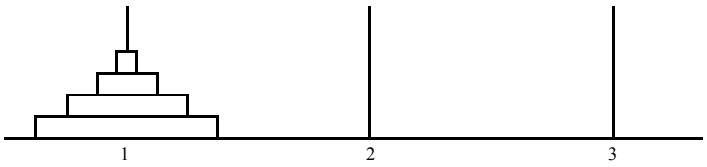
Remarque : Cet exercice est plus simple que le précédent (un seul appel récursif) mais il nécessite d'avoir bien compris les attributs. Il faut réfléchir à l'ordre entre ces 2 exercices ?

Dans cet exercice, mettre en évidence l'utilisation des attributs (qui simplifie la vie !!!). Sans ces attributs une solution pourrait consister à passer en paramètre de la fonction somme un numéro qui mémorise à quel indice du tableau on en est (il faut aussi connaître la taille du tableau).

```
function Somme(T : in Tab) return Float is
begin
   if T'Length = 1 then
      return T(T'First);
   else
      return T(T'First) + Somme(T(T'First+1..T'Last));
   end if;
end Somme;
```

Donner un exemple avec un tableau à 3 ou 4 éléments.
Que se passe-t-il si le tableau contient 0 éléments ?
➔ segmentation fault

6. Tour de Hanoï



Ecrire une procédure qui transfère un nombre de disques N du piquet Origine 1 sur le piquet Destination 3 en utilisant le piquet Auxiliaire 2 comme intermédiaire

```
procedure Deplacer (Nb_Disques : in Integer;  Origine : in Integer;
                                Destination : in Integer;  Auxiliaire : in Integer ) is
begin -- Deplacer
        if Nb_Disques > 0 then -- Encore des disques a transferer?
                -- Transferer Nb_Disques-1 disques sur le piquet Auxiliaire en
                -- utilisant le piquet Destination comme intermediaire
                Deplacer ( Nb_Disques - 1, Origine, Auxiliaire, Destination );
                -- Transferer le disque restant du piquet Origine sur Destination
                Put ("Deplacer le disque restant du piquet");
                Put ( Origine );
                Put (" sur le piquet");
                Put ( Destination );
                New_Line;
                -- Transferer Nb_Disques-1 disques sur le piquet Destination en
                -- utilisant le piquet Origine comme intermediaire
                Deplacer ( Nb_Disques - 1, Auxiliaire, Destination, Origine );
        end if;
end Deplacer;
```

7 Sortie du labyrinthe

Analyse :

On se trouve sur une case . 2 possibilités :

- cette case est une sortie (cas trivial) --> fin
- cette case n'est pas une sortie : on regarde si on peut se déplacer vers une des cases voisines (pas déjà explorée, pas un mur) et on cherche la sortie à partir de cette case voisine --> appel récursif. Si ça marche, c'est terminé (on a une sortie) ; pas besoin d'explorer les autres cases voisins puisqu'on se contente d'un chemin ; si ça n'a rien donné, il faut essayer une autre des cases voisines.

Remarque : pour éviter de boucler en repassant par des cases par lesquelles on est déjà passé, il suffit de marquer les cases par lesquelles on passe.

```
With Ada.text_Io, Ada.integer_Text_IO;
use Ada.text_Io, Ada.integer_Text_IO;

procedure Test_labyrinthe is

type Un_Lab is array (Integer range <>, Integer range <>) of
character;

procedure Lire (Lab : out Un_Lab) is
begin
   for L in Lab'range(1) loop
    for C in Lab'range(2) loop
       get(Lab(L,C));
    end loop;
    skip_line;
    end loop;
end Lire;

procedure Afficher (Lab : in Un_Lab) is
begin
   for L in Lab'range(1) loop
    for C in Lab'range(2) loop
       put(Lab(L,C));
    end loop;
    new_line;
    end loop;
```

```
    new_line(2);
end Afficher;

procedure Trouver_sortie ( Lab : in out Un_Lab; L,C : in Natural;
Trouvee : in out boolean) is

begin
   if L= lab'first(1) or L=Lab'last(1) or C= lab'first(2) or
C=Lab'last(2) then  -- cas trivial
      Trouvee :=True;
   else
      Lab(L,C):='o'; -- marquage
      -- est ?
      if Lab(L,C+1) = '.' then
         Trouver_sortie (Lab, L, C+1, Trouvee) ;
      end if;
      -- nord
      if not Trouvee and Lab(L-1,C) = '.' then
         Trouver_sortie (Lab, L-1, C, Trouvee) ;
      end if;

      -- ouest
      if not Trouvee and Lab(L,C-1) = '.' then
         Trouver_sortie (Lab, L, C-1, Trouvee) ;
      end if;
      -- sud
      if not Trouvee and Lab(L+1,C) = '.' then
         Trouver_sortie (Lab, L+1, C, Trouvee) ;
      end if;
   end if;
   if trouvee then
   -- marquage du chemin
   Lab(L,C) := '*';
   end if;
end Trouver_Sortie;
```

```
Mon_Lab : Un_Lab (1..7, 1..7):= (('H','H','H','H','H','H','H'),
                                 ('H','.','.','H','.','.','H'),
                                 ('H','H','H','H','.','H','H'),
                                 ('H','.','.','.','.','H','H'),
                                 ('H','.','.','H','.','H','H'),
                                 ('H','.','.','.','.','.','H'),
                                 ('H','H','H','H','H','.','H'));

LI,CI : Natural;
OK : boolean :=False;

begin
 --Lire (Mon_Lab);
 Afficher (Mon_Lab);
 put (" Position initiale");
 get(Li); get(CI);
 Trouver_sortie (Mon_Lab, LI, CI, OK);
 if OK then
  afficher (Mon_Lab);
 else
  put_line ("pas de sortie");
 end if;

end Test_labyrinthe ;
```