Récursivité

NF01

Philippe TRIGANO

Définition

- **Une fonction ou une procédure est dite récursive s'il est fait appel à cette fonction ou à cette procédure dans le corps d'instructions qui la définit.**

- **En d'autres termes , la fonction (ou la procédure) s'appelle elle-même**

Exemple

- **Fonction factorielle**
  - n! = n*(n-1)* 1
- **Mais aussi**
  - n! =n*(n-1)!

```
function factorielle(n:integer): longint;
begin
    if (n > 1) then
            factorielle:= n * factorielle (n-1)
    else
            factorielle:= 1;
end;
```

Somme des n premiers nombres

- **Fonction itérative :**

```
function somme(N:integer):longint;
var
    i : integer;
    sum : longint;
begin
    sum:=0;
    for i := 1 to N do sum := sum + i;
    somme := sum;
end;
```

Somme des n premiers nombres

- **Fonction récursive**

  $S(n) = S(n - 1) + n$

  $S(0) = 0$ pour la condition d'arrêt

```
program SommePremiersEntiers ;
var
    n:integer;

function somme ( n : integer ) : integer ;
begin
    if n = 0  then  somme:= 0
              else  somme:= somme (n - 1) + n;
end;

begin
    readln(n);
    writeln(somme (n));
end.
```

```
somme(4)
début
  appel de somme avec n = 3 {début de l'empilement}
    début
        appel de somme avec n = 2
            début
                appel de somme avec n = 1
                    début
                        appel de somme avec n = 0 {fin de l'empilement}
                            début
                                somme ← 0
                            fin {début du dépilement}
                        somme ← somme (0) + 1 = 0 + 1 = 1
                    fin
                somme ← somme (1) + 2 = 1 + 2 = 3
            fin
        somme ← somme (2) + 3 = 3 + 3 = 6
    fin
 somme ← somme (3) + 4 = 6 + 4 = 10
Fin {fin du dépilement}

Résultat : somme (4) = 10
Fin
```

```
program A_Trouver ;
  const
        POINT = '.' ;

    procedure Faire ;
    var
       car : char ;
    begin
       read (car) ;
       if car <> POINT then
                  Faire ;
       write (car) ;
    end;
```

{ programme principal }

```
begin
      writeln ('Entrez  un texte') ;
      Faire ;
      writeln;
end.
```

Exécution :

```
Bonjour.
.ruojnoB
```

Exercice

- **Ecrire une fonction récursive permettant de calculer cos(x) et sin(x)**

- **Principe ?**
  - Exprimer en fonction du cosinus et/ou du sinus d'un nb plus petit

- **Condition d'arrêt ?**

Algorithme

- **sinus**

  Si x est petit alors

  sin(x) ← x

  Sinon

  sin(x) ← 2 sin(x/2) * cos(x/2)

- **cosinus**

  Si x est petit

  cos(x) ← 1

  Sinon

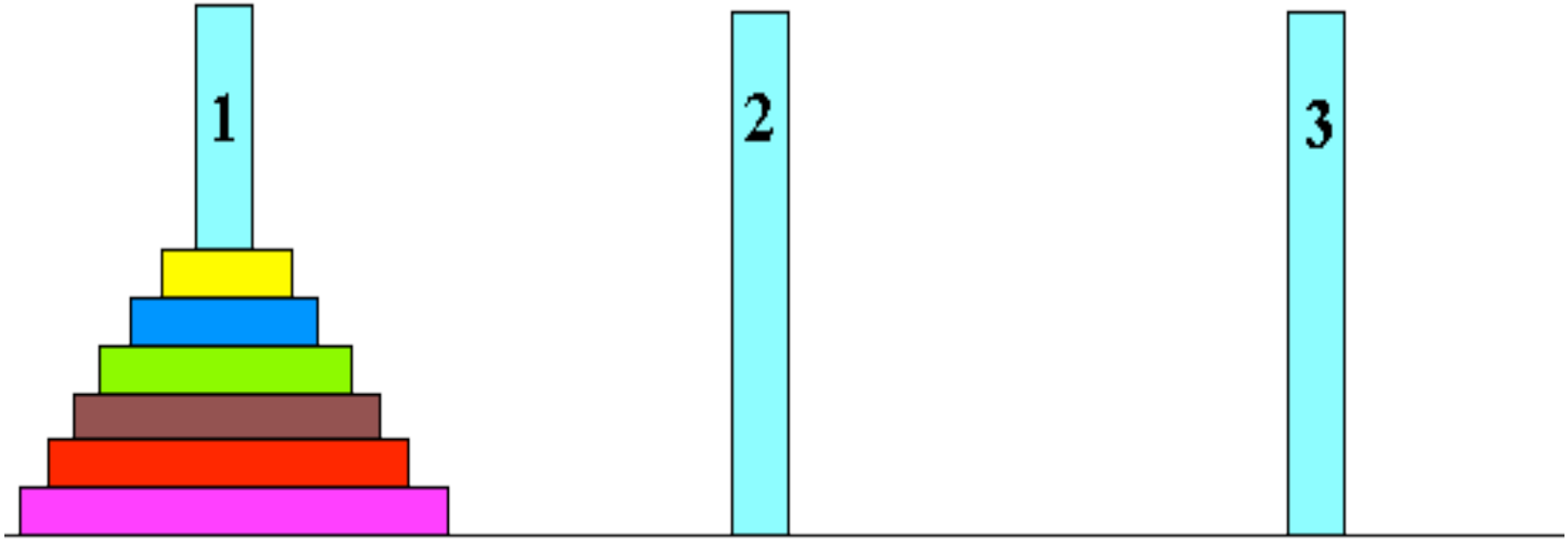
  $\cos(x)=\cos^2(x/2) + \sin^2(x/2)$

```
function cosinus(x: real):real; forward;
  {voir plus loin}

  function sinus(x: real):real;
  begin
    if x <= 0.00001 then
      sinus := x
    else
      sinus := 2 * sinus(x/2) * cosinus(x/2)
  end;

  function cosinus(x: real):real;
  begin
    if x <= 0.00001 then
     cosinus := 1
    else
     cosinus := sqr(cosinus(x/2))-sqr(sinus(x/2))
  end;
```

Tours de Hanoï



BUT ***:*** Déplacer la pile de la tour 1 à la tour 3, en ne déplaçant qu'un disque à la fois, et en s'assurant qu'aucun disque ne repose sur un disque de plus petite dimension.

Algorithme

- **Hypothèse :** on veut **déplacer la pile des n-1 premiers disques** de la tour 1 à la tour 2
- **Action : Déplacer le dernier disque** de la tour 1 à la tour 3
- **Ensuite :** il ne reste plus qu'à **déplacer les n-1 disques** de la tour 2 à la tour 3
- **Condition d'arrêt : nb de disques = 0 (il n'y a plus de disque à déplacer)**

Ecriture en Pascal

```
procedure Hanoi (nbDisques ,T_orig,T_dest,T_inter: integer);
begin
   if (nbDisques > 0)  then
   begin
     Hanoi (nbDisques - 1 , T_orig, T_inter, T_dest) ;
     writeln ('Déplacer le disque de ', T_orig,' à ', T_dest);
     Hanoi (nbDisques - 1 , T_inter, T_dest, T_orig) ;
   end;
end;
```