Les algorithmes récurrents

I. Introduction

Un algorithme récurrent est un algorithme utilisant un traitement répétitif pour produire un résultat calculé. Ce résultat peut dépendre des p résultats précédents : c'est un algorithme récurrent d'ordre p.

Si p = 1, on parle d'algorithme récurrent d'ordre 1

Si p = 2, on parle d'algorithme récurrent d'ordre 2

Exemples :

1) $\forall k \in N^*, n^k = n^{k-1} \times n$ algorithme récurrent d'ordre 1

2) $U_{n+2} = U_{n+1} + U_n$ algorithme récurrent d'ordre 2

II. Calcul de somme

Le calcul de la somme d'une série de nombre est un traitement très utile en programmation. C'est un traitement récurrent d'ordre 1.

Application1 :

Analyser puis écrire un algorithme d'une fonction qui permet de calculer la somme de **n** premiers entiers.

Spécification de la fonction somme :

Résultat : somme

Traitement : pour calculer la somme de n premiers entiers, il suffit d'initialiser la somme à **0** puis ajouter au fur et à mesure les valeurs jusqu'à l'arrivée à **n**

Donc c'est un traitement itératif complet

D'où l'algorithme de la fonction Somme :

0) Fonction Somme (n : entier) : entier
1) Somme ← 0
2) Pour k de 1 à n Faire
    Somme ← Somme + k
   Fin Pour
3) Fin Somme

Tableau de déclaration des objets locaux

Objet	Type/Nature	Rôle
k	Entier	Compteur

Remarque :

Pour une valeur donnée de compteur k, la valeur de la somme est égale à la valeur ancienne incrémenté de la valeur de k.

Somme ← Somme + k

C'est un algorithme récurrent **d'ordre 1**

Activité Pratique : appeler la fonction précédente dans un programme pascal.

III. Recherche du minimum et de maximum

Application1 :

Analyser puis écrire un algorithme d'une procédure qui permet de rechercher le maximum et le minimum dans un tableau T de n entiers.

Spécification de la procédure min_max:

Résultat : min et max

Traitement : on commence par initialiser min et max à T[1], puis parcourir le reste du tableau et comparer chaque fois T[i] à min et à max.

Donc c'est un traitement itératif.

D'où l'algorithme de la procédure min_max :

```
0)  Procédure min_max (t : tab ; n : entier ; var Min, Max : entier)
    1)  Min ← t[1] ; Max ← Min
2)  Pour i de 2 à n Faire
            Si Min < t[i] Alors Min ←t[i]
                Sinon Si Max > t[i] Alors Max ← t[i]
            Fin Si
      Fin Pour
3)  Fin min_max
```

Question : appeler cette procédure dans un programme pascal

Tableau de déclaration des objets locaux

Objet	Type/Nature	Rôle
i	Entier	Compteur

IV. Calcul sur les suites

Application 1 :

Soit $(S_n)_{n\in N}$ la suite des sommes partielles de la série harmonique. C'est à dire pour tout entier n non nul : $S_n = \sum_{k=1}^{n} \frac{1}{k}$

Cette suite est définie par la relation :

$$\begin{cases} S_0 = 0 \\ S_k = S_{k-1} + \frac{1}{k} \end{cases} \qquad \forall\ k \in N^*$$

Question :

Analyser puis écrire l'algorithme d'une procédure suite qui permet de chercher et d'afficher les valeurs successives de cette suite.

Spécification de la procédure suite :

Résultat : afficher les termes de la suite

Traitement :

Pour afficher les **n** premiers termes de la suite, on doit calculer S_1, S_2, S_3, …, S_{n-1}

Donc il s'agit d'un traitement itératif. La valeur d'un terme dépend de la valeur du terme précédent : c'est un algorithme récurrent d'ordre 1

D'où l'algorithme de la procédure suite :

0) Procédure suite (n : entier)
1) S ← 0
2) Ecrire ("Le premier terme = ", S)
3) Pour k de 1 à n Faire

   S ← S + $\frac{1}{k}$

   Ecrire ("Le terme n°", i, " = ", S)

   Fin Pour
4) Fin suite

```
Program terme_suite ;
Uses wincrt ;
Var n: integer;
procedure suite (n:integer);
var k:integer;
   s:real;
begin
    S :=0 ; writeln('S0 = ',s:2:2);
    For k :=1 to n do
       Begin
             S:=S + 1/k;
             Writeln('S',k,' = ',S:2:2);
       End; end;
begin
    repeat
    writeln('enter la valeur de n:'); read(n);
    until n>0;
    suite(n);
End.
```

Application 2 :

Soit $(S_n)_{n\in N}$ la suite des sommes partielles de la série harmonique. C'est à dire pour tout entier n non nul : $S_n = \sum_{k=1}^{n} \frac{1}{k}$

Cette suite est définie par la relation :

$$\begin{cases} S_0 = 0 \\ S_k = S_{k-1} + \frac{1}{k} \end{cases} \qquad \forall\ k \in N^*$$

On veut écrire un programme permettant d'afficher la première valeur de **n** pour laquelle la suite **S_n >= 15**

Spécification de la procédure suite :

Résultat : afficher valeur de n

Traitement :

Pour afficher la première valeur de **n**, on utilise une variable **S** qu'on initialise à **0** et qui permet de stocker les valeurs successives de la suite.

Le nombre d'itération n'est pas connu à l'avance, donc on emploi une structure itérative à condition d'arrêt. De plus la variable **S** est initialisée à **0**, donc on doit passer au moins une fois dans la boucle pour que le résultat soit **>=15**. On peut donc utiliser la boucle **Répéter**.

D'où l'algorithme de la procédure suite :

0) Procédure suite (var k : entier long)
1) K ←0
2) S ←0
3) Répéter
   k ← k + 1
   S ← S + 1 / k
   Jusqu'à S >= 15
4) Ecrire ("La valeur de n est ", K)
5) Fin suite

Application 3 :

Soient a et b deux réels supérieurs ou égaux à 1. On considère la suite numérique $(U_n)_{n \in N}$ définie par :

$U_0 = a, U_1 = b$ Et pour tout entier naturel n, $U_{n+2} = \sqrt{U_n} + \sqrt{U_{n+1}}$

Ecrire le programme d'une procédure qui permet de calculer et d'afficher la valeur de U_n pour des valeurs de a et b supérieure ou égales à 1 et pour n >=2

Spécification de la procédure calcul_uite:

Résultat : afficher la valeur de U_n

Traitement :

La suite (U_n) est récurrente d'ordre 2, le calcul de ses termes nécessite donc d'introduire trois variables :

- U : pour stocker le dernier terme calculé (paramètre de la procédure)
- V : pour stocker le terme précédent
- Aux : permettant à chaque itération de stocker le nouveau terme calculé avant d'effectuer les échanges nécessaires entre U et V

La valeur n étant donnée par l'utilisateur, on connaît à l'avance le nombre d'itérations. On emploie donc une boucle Pour … Faire

On initialise U et V par : U ← a (U1) et V ← b (U0)

D'où l'algorithme de la procédure calcul_suite

```
0) Procédure calcul_suite (var U : entier ; n, a, b : entier)
1) U ← b
2) V ← a
3) Pour k de 2 à n Faire
       Aux ← Racine Carrée (U) + Racine Carrée (V)
       V ← U
       U ← Aux
   Fin Pour
4) Ecrire ("U", n, " = ", U) ;
5) Fin calcul_suite
```

```
program calcul;
uses wincrt;
var a,b,n:integer;
   u:real;
procedure calcul_suite (var u:real);
       var k:integer;v,aux:real;
begin
       u:=a;
       v:=b;
       for k:=2 to n do
       begin
       aux:=sqrt(u)+sqrt(v); {calcul de U(k)}
       v:=u; {V reçoit U(k-1)}
       u:=aux; {U reçoit U(k)}
end; writeln('U',n,'=',u:3:2);
end;
begin
       repeat
       write('a=');read(a);
       write('b=');read(b);
       write('n=');read(n);
       until ((a>=1)and(b>=1)and(n>=2));
       calcul_suite(u);
end.
```

V.Triangle de pascal

Application :

Blaise Pascal a donné son nom à un langage informatique, mais il a aussi introduit les coefficients binomiaux. Pour tout couple d'entiers naturels (n, p), on pose :

$$\binom{n}{p} = \begin{cases} 0 & \text{Si } p > n \\ \dfrac{n!}{(n-p)!\,p!} & \text{Si } p <= n \end{cases}$$

Ces nombres peuvent être représentés dans le triangle de pascal, ils sont reliés les uns aux autres par la formule du triangle :

Pour n ≥1 et p ≥ 1, $\binom{n}{p} = \binom{n-1}{p} + \binom{n-1}{p-1}$

On définit donc un tableau **C** à deux dimensions pour stocker les coefficients du binôme. On les calcule ligne par ligne et on affiche chaque ligne au fur et à mesure.

Ecrire le programme d'une procédure qui permet de remplir et d'afficher le triangle de pascal

Pour tout entiers naturels **n ≥1** et **p ≥ 1,**

Spécification de la procédure Triangle_Triangle

Résultat : remplir et afficher le triangle de pascal

Traitement :

Nous savons que les coefficients sont nuls pour **p>n**. il faut donc initialiser **C[n, p]** à **0** pour tous les couples **(n, p)** vérifiant **0<=n<nmax et n<p<=nmax** .

On emploie donc deux boucles imbriquées :

```
Pour n de 0 à nmax-1 Faire
    Pour p de n+1 à nmax Faire
        C[n, p] ← 0
    Fin Pour
Fin Pour
```

De même, on doit initialiser la colonne C[n, 0] pour tout n à 1:

```
Pour n de 0 à nmax faire
    C[n, 0] ← 1
Fin Pour
```

Pour effectuer les calculs ligne par ligne :

Pour p de 1 à n Faire

C[n, p] ← C[n-1, p] + C[n-1, p-1]

Fin Pour

Pour afficher au fur et à mesure les coefficients calculés, il suffit de faire répéter également l'affichage : Ecrire (C[n, p]," ")

D'où l'algorithme de la procédure demandée :

```
0) Procédure Triangle_Pascal (var C : matrice ; n, p : entier)
1) Pour n de 0 à nmax-1 Faire
       Pour p de n+1 à nmax Faire
           C[n, p] ← 0
       Fin Pour                                initialisation
   Fin Pour
2) Pour n de 0 à nmax Faire
       C[n, 0] ← 1
   Fin Pour
3) Pour n de 1 à nmax Faire
       Ecrire (C[n, 0]," ")
       Pour p de 1 à nmax Faire
           C[n, p] ← C[n -1, p] + C[n -1, p -1]        Ligne n
           Ecrire (C[n, p]," ")
       Fin Pour
   Fin Pour
4) Fin Triangle_Pascal
```