Chapitre III

Les sous programmes

Durée : 12 Heures
Type : Théorique et pratique

I - Partie rappel

I.1 Analyse modulaire :

Pour résoudre un problème complexe, on peut procéder à une décomposition de ce dernier en sous problèmes. Ces derniers sont à leur tour décomposés selon le besoin. La décomposition s'arrête aux sous problèmes relativement simples à résoudre. On associe à chaque sous problème un module (un sous programme) assurant sa résolution.

(Sources livre 2 TI – Chapitre 12)

I.2 Notions de sous-programme :

- Un sous-programme est un ensemble d'instructions, analogue à un programme.
- Sa peut être une procédure ou une fonction.
- Un sous-programme peut être exécuté plusieurs fois grâce à, des appels.
- Une procédure est un sous-programme qui produit zéro ou plusieurs résultats alors qu'une fonction est un sous-programme qui ne produit qu'un seul résultat de type simple.

I.3 Intérêts de l'analyse modulaire :

- Organisation du code source, il est plus efficace de séparer les différentes parties d'un programme.
- La disposition en modules nous permet aussi de savoir lequel des sous programmes est à corriger dans le cas où on a une erreur.
- Il est aussi plus facile de faire évoluer le programme et de passer d'une version à une autre.
- La réutilisation du code.

I.4 Les fonctions

a. Définition

Une fonction est un sous-programme qui retourne un résultat de **type simple** contenu dans son identificateur.

b. Appel d'une fonction

Une fonction se comporte comme une variable. L'appel de la fonction doit nécessairement apparaître dans une expression (d'affectation ou d'affichage,…).

En Algorithme

Variable ← **nom_fonction** (paramètres effectifs)
Ou
Ecrire (**nom_fonction** (paramètres effectifs))

En Pascal

Variable := **nom_fonction** (paramètres effectifs)
Ou
Write (**nom_fonction** (paramètres effectifs))

c. Définition d'une fonction :

En Algorithme

0) **Fonction** nom_fonction (liste des paramètres formels) : Type du résultat ;
1) Instruction 1
2) Instruction 2
3) .....
4) nom_fonction ← résultat ;
5) **Fin** nom_fonction

TDO Locaux

Objet	Type	Rôle

En Pascal

```
Function nom_fonction (liste des paramètres formels) : Type du résultat ;
        Var
            {Déclarations des variables locales} ;
        begin
            {Instructions de la fonction} ;
            Nom_fonction := résultat ;
end;
```

I.5 Les procédures

a. Définition

Une procédure est un sous-programme qui peut retourner zéro ou plusieurs résultats.

b. Appel d'une procédure

Voir livre page 110 (II.2)

c. Définition d'une procédure:

Voir livre page 111 (II.4)

I.6 Déclarations et accès aux objets :

a. Les objets locaux :

Un objet Local est un objet déclaré et connu seulement à l'intérieur d'un sous-programme.

b. Les objets globaux :

Un objet Global est déclaré dans la partie déclarative du programme principal. Il peut être utilisé par le programme principal ou par les différents autres sous-programmes.

c. Accès aux objets

La porté de l'objet défini les possibilités d'accès à ce dernier à partir de différents endroits du programme. Exemple, un objet déclaré dans un sous-programme n'est pas accessible à partir du programme principal. Par contre, un objet global est accessible à partir d'un sous-programme.

I.7 Les paramètres et leurs modes de passage :

a. Les paramètres formels et les paramètres effectifs :

- Les paramètres formels figurent dans l'entête de la définition d'un sous-programme (fonction ou procédure).
- Les paramètres effectifs figurent dans l'appel d'un sous-programme (fonction ou procédure).

Remarques :

- Les paramètres **effectifs** et les paramètres **formels** doivent s'accorder du point de vue nombre et ordre.
- Leurs types doivent être identiques ou compatibles, selon le mode de passage des paramètres.

b. Passage de paramètre par valeur et par variable :

La substitution des paramètres effectifs aux paramètres formels s'appelle passage de paramètre, il s'agit en fait de transfert de données entre le programme principal (P.P) et le sous programme ou l'inverse.

Nous utiliserons deux modes de passage de paramètres :

1. Le passage de paramètres par **valeur**
- Le passage de paramètres par **variable**

Si le paramètre formel n'est pas précédé par le mot **VAR** alors il s'agit d'un passage de paramètres effectifs par **valeur.**

Si le paramètre formel est précédé par le mot **VAR** alors il s'agit d'un passage de paramètres effectifs par **variable.**

Voir livre page 110 (II.3)

{exemple d'un programme construit en modules}	Commentaire

```
Program Combinaison ;
Uses wincrt;

Var
      f1, f2, f3 ,p , n : integer ;
      c : integer ;
```

Déclaration du programme principal

```
Procedure saisie ( var n, p :integer);
Begin
Repeat
      Writeln('donner un entire n ');
      readln(n) ;
      Writeln('donner un entire p');
      readln(p) ;
until (n>=p) and ( p>0);
end;
```

Entête

Le corps de la procédure

Sous programme Appelé

Paramètres formels

```
Function fact (x: integer) : integer ;
Var f, c :integer;
Begin
   f:=1;
 For c :=1  to  n  Do
     Begin
           f:= f * c;
       End;
Fact:=f;
End ;
```

Entête

Déclaration

Le corps de la fonction

Sous programme Appelé

```
Begin
Saisie (n, p);  {Appel}

f1:= fact (n);   {Appel}
f2 := fact (p);   {Appel}
f3 := fact (n-p);  {Appel}

c := f1 div (f2 *f3) ;

writeln ('la combinaison de p objets parmi n est  =  ', c);
End.
```

Paramètres effectifs

Programme principal

II - Applications

- **Application 1 :**

20	23	21	22	26	27	23

Ecart type

$$\text{Moyenne} = (\sum_{i=1}^{n} Mi) / n \qquad \text{écart type} = \sqrt{\sum_{i=1}^{n} (Mi - Moy)^2}$$

- **Application 2 :**

Énoncé :

On désire vérifier l'existence d'une chaîne de caractère **ch** dans un tableau **T** de n chaînes de caractères ($2 \leq n \leq 10$). Faire l'analyse de se problème, tout en prévoyant un module pour la saisie et un module pour la vérification de l'existence de **ch** dans **T**. Sachant que ce module renvoi l'indice de la case dans laquelle **ch** a été trouvé, sinon zéro pour dire que ch n'existe pas dans T afficher un message indiquant le résultat de la vérification.

Exemple :

ch = "BAC"

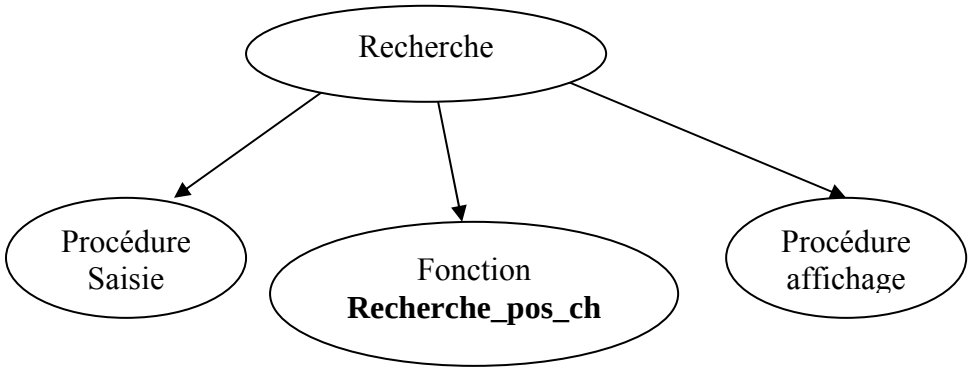
T =

INFO	BAC	SPORT

Affichage : ch existe à la case 2.

Solution :

a. Découpage modulaire



b. Analyse principale :

Résultat : affichage (p)

Traitements :

p ← Recheche_pos_ch (ch, T, n)

(ch, T, n) = saisie (ch, T, n)

Algorithme :

0) Début Recherche
1) saisie (ch, T, n)
2) p ← Recheche_pos_ch (ch, T, n)
3) affichage (p)
4) Fin Recherche

T.D.O Globaux

Objet	Type	Rôle
p, n	Octet	
ch	chaîne	
T	Tch	

Type
Tch = tableau de 10 chaînes

c. Analyse procédure affichage :

Procédure **affichage** (p : octet)

Résultat : Affichage
Traitements :

```
Si  p = 0  alors écrire ("La chaîne n'existe pas dans T")
           Sinon écrire ("La chaîne existe dans T à la case ", p)
Fin Si
```

Algorithme :

```
0) Procédure affichage ( p : octet)
1) Si  p = 0  alors écrire ("La chaîne ",ch," n'existe pas dans T")
               Sinon écrire ("La chaîne ",ch," existe dans T à la case ",p)
   Fin Si
2) Fin affichage
```

d. Analyse procédure affichage :

Fonction **Recheche_pos_ch** (ch : chaîne, T : Tch, n : octet) : octet

Résultat : Recheche_pos_ch
Traitements :

```
Recheche_pos_ch ← p
i ← 1
existe ← faux
Répéter
        Si T[i] = ch alors   p ← i
                             existe ← vrai
                     sinon   i ← i + 1
        Fin Si
Jusqu'à (existe) ou (i > n)
```

Algorithme :

```
0) Fonction Recheche_pos_ch ( ch : chaîne, T : Tch, n : octet) : octet
1) p ← 0
2) i ← 1
   existe ← faux
   Répéter
        Si T[i] = ch alors   p ← i
                             existe ← vrai
                     sinon   i ← i + 1
        Fin Si
   Jusqu'à (existe) ou (i > n)
3) Recheche_pos_ch ← p
4) Fin Recheche_pos_ch
```

T.D.O Locaux

Objet	Type	Rôle
i	Octet	compteur
existe	Booléen	drapeau
p	Octet	

e. Analyse de la procédure saisie :

Procédure **saisie** (var ch : chaîne, var T : Tch, var n : octet)

Résultat : ch, T et n saisis
Traitements :

```
Ch = donnée
Répéter
        n  = donnée (" Donner n entre 2 et 20 : " )
Jusqu'à  n dans [2..20]
Pour i de 1 à n faire
        Ecrire (" Donner la case ",i, " : "), Lire (T[i])
Fin Pour
```

Algorithme :

0) Procédure **saisie** (var ch : chaîne, var T : Tch, var n : octet)
1) Ecrire ("Donner une chaîne : "), Lire (ch)
2) Répéter
        Ecrire (" Donner n entre 2 et 20 : "), lire (n)
   Jusqu'à n dans [2..20]
3) Pour i de 1 à n faire
        Ecrire (" Donner la case ",i, " : "), Lire (T[i])
   Fin Pour
4) Fin **saisie**

T.D.O Locaux

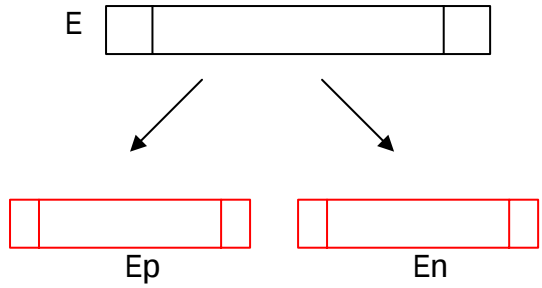
Objet	Type	Rôle
i	Octet	compteur

f. Traduction Pascal : (voir fichier : Pos_ch_t.pas)

- **Application 3 :**

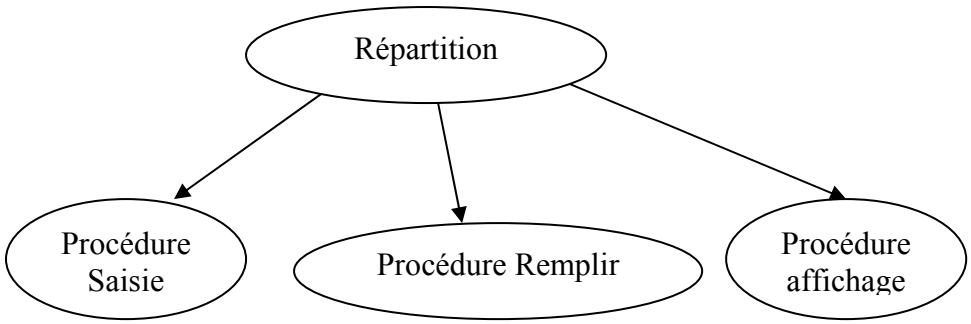
Énoncé :

Ecrire un programme nommé "rangement", qui permet de saisir un tableau E de n entier (1≤ n ≤ 50), et réaffecter les éléments positifs de E dans le tableau Ep (de taille n). (Zéro étant considéré comme positif) de même pour les éléments négatifs. Les éléments de E sont des entiers de 3 chiffres.



Solution :

a. Découpage modulaire



b. Analyse principale :

Résultat : affichage (Ep, En, n)

Traitements :

Remplir (Ep, En, E, n)

Saisie (E, n)

Algorithme :

0) Début répartition
1) Saisie (E, n)
2) Remplir (Ep, En, E, n)
3) affichage (Ep, En, n)
4) Fin répartition

T.D.O Globaux

Objet	Type	Rôle
n	Octet	
E	Elément	
Ep	Elément	
En	Elément	

Type
Elément = tableau de 50 entiers

c. Analyse procédure affichage :

Procédure **affichage** (Ep, En : élément, n : octet)

Résultat : Affichage
Traitements :
Prévoir deux boucle pour permettant l'affichage des 2 tableaux Ep et En.

Algorithme :

0) Procédure **affichage** (Ep, En : élément, n : octet)
1) Pour i de 1 à n faire
   Ecrire (Ep[i])
   Fin pour
2) Pour i de 1 à n faire
   Ecrire (En[i])
   Fin pour
3) Fin **affichage**

T.D.O Locaux

Objet	Type	Rôle
i	octet	Compteur

d. Analyse procédure remplir :

Procédure **remplir** (var Ep, En : élément, E : élément, n : octet)

Résultat : Ep, En remplis
Traitements :
Prévoir une boucle pour qui permet d'effectuer un parcours total dans E.
Prévoir 3 compteurs ,
i pour avancer dans E
j pour avancer dans Ep
k pour avancer dans En

Initialiser j, k à 1
Pour chaque case de E faire le test :
Si E[i] > 0 alors Ep[j] ←E[i]
j ← j + 1
Sinon En[k] ← E[i]
k ← k + 1
Fin si

Algorithme :

```
0) Procédure remplir (var Ep, En : élément, E : élément, n : octet)
1) K ← 1
   J ← 1
   Pour i de 1 à n faire
        Si E[i] > 0 alors Ep[j] ← E[i]
                          j ← j + 1
                     Sinon En[k] ← E[i]
                          k ← k + 1
        Fin si
   Fin pour
2) Fin remplir
```

T.D.O Locaux

Objet	Type	Rôle
i	octet	Compteur
j	octet	Compteur
k	octet	Compteur

e. Analyse procédure saisie :

Procédure **saisie** (var E : élément, var n : octet)
Résultat : E, n saisis

Traitements : saisie contrôlée de n [1..50]

Boucle pour jusqu'à n, prévoir une saisie contrôlée sur les éléments de E qui doivent être de 3 chiffres.

Algorithme

```
0) procédure saisie (var E : élément, var n :octet)
1) répéter
        lire (n)
   jusqu'à n dans [1..50]
2) pour i de 1 à n faire
     répéter
        lire (E[i])
     jusqu'à Abs (E[i]) dans [100..999]
   fin pour
3) fin saisie
```

f. Traduction Pascal : (voir fichier : distrib.pas)

- **Application 4**

Exercice 2 page 124

Énoncé :

Soient les suites U et W définies par :

$U_0=A$, $U_1=B$, $Un = (U_{n-1} + U_{n-2})/2$ A et B sont deux entiers données.

$W_0 =A$,

$W_n = U_n - W_{n-1}$

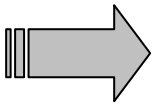
On veut écrire un programme qui permet de calculer la suite W pour un entier n donnée avec $n < 100$.

Solution :

Brouillon : analyse numérique

Pour $U_0 = 3$, $U_1 = -7$ et $n = 4$

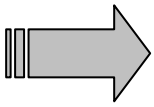
$U_4 = (U_3 + U_2) / 2$ $U_4 = (-4 - 2) / 2 = -3$
$U_3 = (U_2 + U_1) / 2$ $U_3 = (-2 - 7) / 2 = -4$
$U_2 = (U_1 + U_0) / 2$ $U_2 = (-7 + 3) / 2 = -2$



$U_4 = -3$

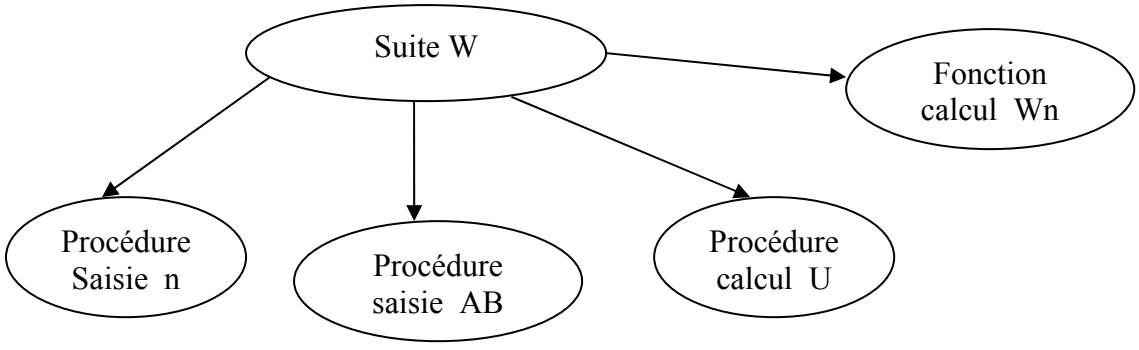
Pour $W_0 = 3$

$W_4 = U_4 - W_3$ $W_4 = -3 + 12 = 9$
$W_3 = U_3 - W_2$ $W_3 = -4 - 8 = -12$
$W_2 = U_2 - W_1$ $W_2 = -2 + 10 = 8$
$W_1 = U_1 - W_0$ $W_1 = -7 - 3 = -10$



$W_4 = 9$

a. Découpage modulaire



b. Analyse principale :

Résultat : écrire ("Wn est égal à : " , calcul_Wn (U,A,n))

Traitements :

calcul_U (U, A, B, n)

saisie_n (n)

saisie_AB (A, B)

Algorithme :

0) Début suiteW
1) saisie_n (n)
2) saisie_AB (A, B)
3) calcul_U (U, A, B, n)
4) écrire ("Wn est égal à : " , calcul_Wn (U,A,n))
5) Fin suiteW

T.D.O Globaux

Objet	Type	Rôle
n	entier	
A, B	entier	
U	Suite	Tableau contenant les termes de U

Type
Suite = tableau de 100 entiers

c. Analyse de la fonction calcul_wn :

Fonction **calcul_Wn** (U: suite, A:entier, n:entier) : entier

Résultat : calcul_Wn

Traitements :

calcul_Wn ← W[i]
Initialiser W[1] avec A
Créer une boucle permettant d'effectuer un parcours total sur W, on calculera la suite w selon la formule :

$W_n = U_n - W_{n-1}$

Algorithme :

0) Fonction **calcul_Wn** (U: suite, A:entier, n:entier) : entier
1) W [1] ← A
2) Pour i de 2 à n+1 faire
   W[i] ← U[i] – W[i-1]
   Fin pour
3) calcul_Wn ← W[i]
4) Fin **calcul_Wn**

T.D.O Locaux

Objet	Type	Rôle
i	entier	Compteur
w	Suite	

d. Analyse de la procédure calcul_U :

Procédure **calcul_U** (var U: suite, A,B : entier, n:entier)

Résultat : Le tableau U rempli

Traitements :

Initialiser U[1] avec A
Initialiser U[2] avec B

Créer une boucle permettant d'effectuer un parcours total sur U, et calculer la suite U selon la formule :

$Un = (U_{n-1} + U_{n-2})/2$

Algorithme :

0) Procédure **calcul_U** (var U: suite, A, B : entier, n:entier)
1) U[1] ← A
2) U[2] ← B

   Pour i de 2 à n+1 faire
   U[i] ← (U[i – 1] + U[i – 2]) div 2
   Fin pour

3) Fin **calcul_U**

T.D.O Locaux

Objet	Type	Rôle
i	entier	Compteur

e. Analyse de la procédure saisie_AB :

Procédure **saisie_AB** (var A, B : entier)

Résultat : A, B saisies
Traitements :

A = donnée
B = donnée

Algorithme :

0) Procédure **saisie_AB** (var A, B : entier)
1) Lire (A)
2) Lire (B)
3) Fin **saisie_AB**

f. Analyse de la procédure saisie_n :

Procédure **saisie_n** (var n : entier)

Résultat : n saisie

Traitements :

Répéter
n = donnée
Jusqu'à (n <100) et (n >1)

Algorithme :

0) Procédure **saisie_n** (var n : entier)
1) Répéter
n = donnée
Jusqu'à (n <100) et (n >1)
2) Fin **saisie_n**

g. Traduction Pascal : (voir fichier : suitew.pas)