

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

وزارة التعليم العالي والبحث العلمي

Université 08 mai 1945

Guelma



جامعة 08 ماي 1945

قالمة

Faculté des mathématiques, d'Informatique et des sciences de la matière.
Département d'informatique.

POLYCOPIE DE TRAVAUX PRATIQUES

Module :

Informatique 2

Destiné aux étudiants en 1^{er} année Licence
Sciences et Technologie.

Elaboré Par

Dr Chokri FERKOUS

Année universitaire : 2020-2021

Préface

Les structures de données constituent un concept vital en informatique, elles présentent le format d'organisation, d'enregistrement et de gestion des données qui permet une manipulation efficaces.

Ce polycopié de travaux pratiques est destiné aux étudiants en première année licence (Tronc commun LMD) assurées au département des sciences et technologie. Il regroupe un certain nombre de concepts de structures de données étudiés dans les différents chapitres du cours d'informatique 2 pendant le deuxième semestre.

Ce document comporte trois chapitres présentant chacun un concept de programmation; chaque chapitre comprend trois sections, dans la première nous fournissons un rappel de cours, puis une section sous la forme (Question/Réponse) est posée, où les réponses sont des parties de codes, cette section permet de renforcer les compétences de l'étudiant à comprendre les notions étudiées ; Dans la section suivante quelques exercices corrigés offrent un moyen de mettre en pratique les connaissances des étudiants et d'améliorer leurs compétences acquises. Enfin dans la dernière section nous terminons par une conclusion.

La première partie de ce polycopié est réservé à la manipulation des structures indicées (tableaux unidimensionnels et bidimensionnels) a travers de quelques exercices, ces concepts représentent les structures de données les plus fondamentales en programmation ; La deuxième partie englobe un rappel de cours et quelques exercices pour découvrir les avantages des procédures et fonctions et leurs principes de base. Enfin, dans la dernière partie nous présentons les enregistrements et leur manipulation à travers les fichiers typés qui nous permettra de stocker les données dans des supports physiques (Disques durs, CDs ...).

L'objectif principal de ces travaux pratiques est de proposer des exercices pratiques et d'implémenter des solutions efficaces qui aident les étudiants à comprendre les concepts abordés. Les solutions proposées dans ce document ont été implémentée en langage Pascal sous l'environnement de développement Lazarus (Version 2.0.6) ; Lazarus est le plus populaire des EDI compatible Delphi basé sur *Free Pascal*, son objectif est de fournir aux programmeurs Pascal un environnement de développement facile à utiliser s'approchant le plus possible de Delphi, il peut être utilisé pour créer des applications en console ou graphiques. *Free Pascal* est un compilateur multiplateforme pour le langage Pascal, il est développé en tant que logiciel libre, il fonctionne sous (Windows, Linux, macOS...ect).

Table des matières

PREFACE	1
TABLE DES MATIERES	2
I. LES VARIABLES INDICEES	4
I.1. RAPPEL (LES TABLEAUX UNIDIMENSIONNELS)	5
I.2. QUESTIONS	6
I.3. EXERCICE 1 : (SOMME ET PRODUIT DES ELEMENTS).....	7
I.4. EXERCICE 2 : (REORGANISATION D'UN TABLEAU).....	8
I.5. EXERCICE 3 : (LA MOYENNE DES NOTES).....	9
I.6. EXERCICE 4 : (TABLEAU DE FIBONACCI)	10
I.7. EXERCICE 5 : (SOUS-TABLEAU).....	10
I.8. EXERCICE 6 : (ELEMENTS CROISSANTS)	12
I.9. EXERCICE 7 : (LA PLUS GRANDE VALEUR D'UN INTERVALLE).....	13
I.10. EXERCICE 8 : (LE PLUS GRAND PGCD).....	14
I.11. EXERCICE 9 : (FUSIONNER DEUX TABLEAUX)	15
I.12. RAPPEL (LES TABLEAUX BIDIMENSIONNELS).....	17
I.13. QUESTIONS :	17
I.14. EXERCICE 10 : (MATRICE BINAIRE)	18
I.15. EXERCICE 11 (CHANGER LES ELEMENTS DE LA LIGNE ET DE LA COLONNE)	20
I.16. EXERCICE 12 (LA LIGNE AYANT LA PLUS GRAND SOMME)	21
I.17. EXERCICE 13 : (UNE SOMME SUPERIEURE OU EGALE A UNE VALEUR)	22
I.18. EXERCICE 14 : (CHAQUE ELEMENT EST REMPLACE PAR LE PLUS PETIT L'ELEMENT A SA DROITE)	24
I.19. EXERCICE 15 : (LES ELEMENTS DE PIC).....	25
I.20. EXERCICE 16 : (LES INDICES D'EQUILIBRES D'UNE MATRICE).....	26
I.21. EXERCICE 17 : (MULTIPLIER DEUX TABLEAUX BIDIMENSIONNELS)	28
I.22. CONCLUSION	30
II. LES PROCEDURES ET FONCTIONS	31
II.1. RAPPEL (LES PROCEDURES ET FONCTIONS).....	32
II.2. EXERCICE 1 : (PLUS PROCHE DE 20).....	34
II.3. EXERCICE 2 : (SOMME / DOUBLE).....	35
II.4. EXERCICE 3 : (APPROXIMATION D'UNE EXPRESSION)	35
II.5. EXERCICE 4 : (LES NOMBRES PREMIERS)	37
II.6. EXERCICE 5 : (CHIFFRE COMMUN)	38
II.7. EXERCICE 6 : (LES NOMBRES PARFAITS)	40
II.8. EXERCICE 7 : (LES MULTIPLES).....	42

II.9.	EXERCICE 8 : (LES VOISINS CROISSANTS D'UN TABLEAU).....	43
II.10.	EXERCICE 9 : (CALCUL MATRICIEL).....	44
II.11.	EXERCICE 10 : (MATRICE EQUILIBREE).....	46
II.12.	EXERCICE 11 : (TABLEAU CENTRE)	49
II.13.	EXERCICE 12 : (DECIMAL / BINAIRE / OCTAL)	51
II.14.	EXERCICE 13 : (LES ANAGRAMMES)	55
II.15.	CONCLUSION	56
III.	LES ENREGISTREMENTS ET FICHIERS	58
III.1.	LES ENREGISTREMENTS ET FICHIERS (RAPPEL)	59
III.2.	QUESTIONS	59
III.3.	EXERCICE 1 : (GESTION DE SCOLARITE)	60
III.4.	EXERCICE 2 : (GESTION DES VENTES AUTOMOBILES).....	66
III.5.	EXERCICE 3 : (GESTION DES CLIENTS)	73
III.6.	EXERCICE 4 : (LES STATISTIQUES)	80
III.7.	CONCLUSION	85
	REFERENCES BIBLIOGRAPHIQUES	86

Chapitre

1

Les variables indicées

I.1. Les tableaux unidimensionnels (Rappel)

Un tableau est une structure qui alloue de manière contiguë des données de même type, cela signifie qu'un tableau permet de stocker les données séquentiellement en mémoire, ce qui facilite l'accès aux éléments du tableau. Chaque élément peut être localisé par son indice.

- ✓ Le nombre d'éléments d'un tableau est un constant défini par les valeurs minimum (`indice_min`) et maximum (`indice_max`), avec `indice_min < indice_max`.
- ✓ La taille du tableau $N = \text{indice_max} - \text{indice_min} + 1$.
- ✓ L'accès aux éléments de la table s'effectue directement par l'indice ;

En Pascal on peut déclarer un Tableau de la manière suivante :

```
Type Tableau = array[indice_min..indice_max] of type;
var T : Tableau;
```

L'accès à l'élément ayant l'indice i d'un tableau T s'effectue par l'appel $T[i]$, tel que les indices des éléments vont de `indice_min` à `indice_max`.

Généralement `indice_min=1`, dans ce cas la déclaration d'un tableau s'effectue de cette manière :

```
Const N=indice_max;
Type Tableau=array[1..N]of integer;
var T:Tableau;
```

Exemple

La Taille du tableau

Type des éléments

```
Const N=9;
Type Tableau=array[1..N]of integer;
var T:Tableau;
```

Les éléments

L'indice

22	15	47	18	41	37	84	16	37
1	2	3	4	5	6	7	8	9

Taille N=9

Conformément à l'illustration précédente, voici les points importants à prendre en compte.

1. L'indice commence par 1.
2. La longueur du tableau =9, ce qui signifie que ce tableau peut stocker 9 éléments.
3. Chaque élément est accessible à travers son indice.

Par exemple, nous pouvons récupérer l'élément à l'index 6 : $T[6]=37$.

I.2. Questions

Comment pouvez-vous demander à l'utilisateur de saisir les éléments d'un tableau de 30 cellules de type réel ?

Comment pouvez-vous afficher les éléments de ce tableau ?

Réponses

```
Program Questions1;
Uses crt;
Const N=8;
Type Tableau=array[1..N]of real;
var T:Tableau;
    i:integer;
begin
    for i:=1 to N do
        begin
            write('Entrer la valeur N°',i,':');
            read(T[i]);
        end;
    for i:=1 to N do
        begin
            write('La valeur N°',i,'=', T[i]);
        end;
    readkey ;
end.
```

The code is enclosed in a rectangular box. Two callout boxes with pointer lines are present. The first callout box, labeled 'L'insertion des éléments', points to the inner loop that reads user input. The second callout box, labeled 'L'affichage des éléments', points to the inner loop that displays the array elements.

L'insertion d'un ou plusieurs éléments dans un tableau est appelée opération d'insertion. Pour chaque itération i un nouvel élément entré par l'utilisateur est ajouté à l'index i du tableau T .

L'opération d'affichage consiste à traverser tous les éléments du tableau un par un, en affichant à chaque itération i l'élément enregistré à l'indice i du tableau T .

I.3. Exercice 1 (Somme et produit des éléments)

Écrire un programme en pascal qui permet d'enregistrer 8 valeurs paires dans un tableau T puis calculer et afficher la somme et le produit des éléments de ce tableau. (Le programme doit prendre en compte les cas d'erreur pendant la saisie des valeurs paires).

Solution :

```
Program ExerciceI_1;
  Uses crt;
  Const N=8;
  Type Tableau=array[1..N]of integer;
  var T:Tableau;
      i,s,p:integer;
begin
  for i:=1 to n do {*L'insertion des n éléments du tableau*
    repeat
      write('Entrer la valeur N ',i,': ');
      readln(T[i]);
      {*Erreur qui s'affiche Si la valeur est impaire*
      if(T[i] mod 2<>0) then writeln('Erreur de saisie!!');
    until (T[i] mod 2=0);
    {calculer et afficher la somme des éléments du tableau*
    s:=0;
    for i:=1 to N do s:=s+T[i];
    writeln('La somme des elements du Tableau = ',s);
    {calculer et afficher le produit des éléments du tableau*
    p:=1;
    for i:=1 to N do p:=p*T[i];
    writeln('Le produit des elements du Tableau = ',p);
    readkey ;
end.
```


I.4. Exercice 2 (Réorganisation d'un tableau)

Ecrivez un programme Pascal qui demande à l'utilisateur de remplir un tableau **T** de 10 entiers, réorganisez ce tableau de sorte que les éléments ayant la même valeur apparaissent ensemble en utilisant un seul tableau, puis afficher le tableau **T**.

Solution :

```
program Exercice1_2;
uses crt;
const n=10;
type Tab=array[1..n]of integer;
var T:Tab;
    i,j,k,v:integer;
begin
for i:=1 to n do {*L'insertion des éléments du tableau T*}
begin
write('Entrer La valeur N: ',i,': ');
readln(T[i]);
end;
for i:=1 to n-1 do {*réorganisation du tableau T*}
begin
j:=i+1;
for k:=i+1 to n do
if (T[i]=T[k]) then
begin
v:=T[k];
T[k]:=T[j];
T[j]:=v;
j:=j+1;
end;
end;
for i:=1 to n do writeln('la valeur N ',i,'= ', T[i]);
readkey;
end.
```

I.5. Exercice 3 (La moyenne des notes)

Ecrire un programme Pascal qui demande à l'utilisateur 9 valeurs réelles correspondant aux notes d'un étudiant (entre 0 et 20), qui les stocke dans un tableau et qui calcule leur moyenne, puis il affiche : "Admis" si la moyenne est supérieure ou égale à 10 ou «Ajourné» dans le cas contraire. (Le programme doit prendre en compte tous les cas d'erreur pendant la saisie des notes)

Solution

```
program Exercice1_3;
  uses crt;
  const n=9;
  type Tab=array[1..n]of real;
  var T:Tab;
      i:integer;
      somme,moyenne:real;
begin
  for i:=1 to n do {*L'insertion des notes dans le tableau*
    repeat
      write('Entrer la note N ',i,': ');
      readln(T[i]);
      if(T[i]<0)or(T[i]>20) then
        writeln('Erreur de saisie!!');
      until (T[i]>=0) and (T[i]<=20);
{*calcul de la somme des éléments du tableau*
    somme:=0;
    for i:=1 to n do somme:=somme+T[i];
    moyenne:=somme/n; {*calcul de la moyenne des notes*
    if(moyenne>=10) then
      writeln('Admis')
    else
      writeln('Ajourné');
  readkey;
end.
```

I.6. Exercice 4 (Tableau de Fibonacci)

Écrire un programme en Pascal qui remplit le tableau **T** de 40 éléments par la séquence de Fibonacci qui se définit comme suit :

$$\begin{cases} T[1] = 1 \\ T[2] = 1 \\ T[i] = T[i-2] + T[i-1] \text{ pour } i > 2 \end{cases}$$

Puis demander à l'utilisateur de saisir une valeur $v \in [1,40]$ et afficher le terme v de Fibonacci à travers le tableau **T**.

Solution

```

program Exercice1_4;
  uses crt;
  const n=40;
  type Tab=array[1..n]of longint;
  var T:Tab;
      i,v:integer;
begin
  T[1]:=1; {*initialisation de la 1ere valeur de Fibonacci *}
  T[2]:=1; {*initialisation de la 2eme valeur de Fibonacci *}
  {*initialisation de la 1ere valeur de Fibonacci *}
  for i:=3 to n do T[i]:=T[i-2]+T[i-1];
  repeat {*une boucle pour assurer que 0<v<=n *}
    write('Entrez le numéro du terme à afficher: ');
    readln(v);
    if (v<=0) or (v>n) then
      writeln('Erreur de saisie!!');
  until (v>0) and (v<=n);
  {*afficher le terme v de Fibonacci *}
  writeln('Le terme ',v,' de Fibonacci =', T[v]);
  readkey;
end.

```

I.7. Exercice 5 (sous-tableau)

Ecrivez un programme qui affiche les sous-tableaux d'un tableau **T** ayant une somme donnée par l'utilisateur.

Solution

```
program Exercice1_5;
  uses crt;
  const n=9;
  type Tab=array[1..n]of real;
  var T:Tab;
      i,j,k:integer; s,somme:real;
begin
  (*L'insertion des n éléments du tableau*)
  for i:=1 to n do readln(T[i]);
  write('Entrer la somme à chercher:');
  readln(somme);
  s:=0;
  i:=1;
  while (i<=n) do (*chercher un sous-tableau à partir de chaque indice i<=n*)
    begin
      j:=i;
      s:=0;
      (*calculer la somme des valeurs du sous-tableau de l'indice i à l'indice
      j<=n telque cette somme < somme*)
      while (s<somme) and (j<=n) do
        begin
          s:=s+T[j];
          j:=j+1;
        end;
      (*si la somme des valeurs du sous-tableau de l'indice i à l'indice j
      =somme alors afficher ce sous-tableau*)
      if (s=somme) then
        for k:=i to j-1 do write(T[k]:4:2,' ');
        writeln;
        i:=i+1;
      end;
    readkey;
  end.
```

I.8. Exercice 6 (éléments croissants)

Ecrivez un programme en Pascal qui demande à l'utilisateur de remplir un tableau de 5 entiers compris entre 0 et 8 ; et qui permet qui vérifier si les éléments de ce tableau sont strictement croissants.

Solution :

```
program Exercice1_6;
uses crt;
const n=5;
type Tab=array[1..n]of integer;
var T:Tab;
    i:integer;
    trier:boolean;
begin
  for i:=1 to n do {L'insertion des 5 éléments du tableau*}
    repeat {forcer l'utilisateur à saisir une valeur 0≤T[i]≤8*}
      write('Entrer la valeur N ',i,': ');
      readln(T[i]);
      if(T[i]<0) or (T[i]>8) then
        writeln('Erreur de saisie!!');
    until (T[i]>=0) and (T[i]<=8);
  i:=1;
  trier:=true; {nous supposons que ce tableau est trié*}
  while(trier and (i<n))do
    begin
      {le tableau est considéré trié si la valeur de l'indice i T[i]≤T[i+1]*}
      if(T[i]>=T[i+1]) then trier:=false;
      i:=i+1;
    end;
  if(trier) then write('les éléments en ordre croissant')
  else write('les éléments ne sont pas en ordre croissant');
  readkey;
end.
```

I.9. Exercice 7 : (La plus grande valeur d'un intervalle)

Ecrivez un programme en Pascal qui demande à l'utilisateur de remplir un tableau de 10 entiers et qui permet de trouver et afficher la plus grande valeur entre 10 et 20 parmi les 10 éléments du tableau, le programme doit afficher « aucune des 10 valeurs n'est dans cette intervalle » si toutes les valeurs de ce tableau sont en dehors de l'intervalle [10, 20].

Solution :

```

program Exercice1_7;
Uses crt;
const n=10;
type Tab=array[1..n]of integer;
var T:Tab; i,max:integer;
    exist : boolean ; {*variable boolean pour verifier la présence
                        d'une valeur comprise entre 10 et 20*}
begin
    for i:=1 to n do readln(T[i]);
    exist=false ; {*aucune valeur comprise entre 10 et 20*}
    for i:=1 to n do {*pour chaque element T[i] du tableau*}
        begin
            {*tester si la valeur T[i] est comprise entre 10 et 20*}
            if(T[i]>=10) and (T[i]<=20) then
                if(not exist) then
                    begin
                        exist:=true;
                        max:=T[i]; {*initialiser max*}
                    end
                else if(T[i]>max) then max:=T[i];
            end;
        end;
    if(exist) then writeln(max) {*afficher max si exist=true*}
    else
        writeln('aucune des 10 valeurs n'est dans
                l''intervalle [10, 20]');
    readkey ;
end.

```

Écraser la valeur max s'il existe une valeur comprise entre 10 et 20 trouvée précédemment

Chercher la 1ère valeur comprise entre 10 et 20 trouvée

I.10. Exercice 8 (Le plus grand PGCD)

Ecrire un programme qui demande à l'utilisateur de saisir 9 entiers, qui les stockent dans un tableau **T**, puis calculer et afficher le PGCD (Plus Grand Commun Diviseur) de tous les paires des 9 éléments de ce tableau; finalement le programme doit afficher le plus grand PGCD.

Solution :

```

program Exercice1_8;
uses crt;
const n=9;
type Tab=array[1..n]of integer;
var T:Tab;
    i,j,v1,v2,pgcd,pgcdMax:integer;
begin
  for i:=1 to n do {L'insertion des 9 éléments du tableau*}
    begin
      write('Entrer La valeur N: ',i,': ');
      readln(T[i]);
    end;
  pgcdMax :=1 ; {initialiser le plus grand PGCD par 1*}
  for i:=1 to n-1 do
    for j:=i+1 to n do
      begin
        v1:=T[i]; v2:=T[j];
        repeat {chercher le PGCD du (T[i] et T[j])*}
          if(v1>v2) then v1:=v1-v2
          else if(v2>v1) then v2:=v2-v1;
        until (v1=v2);
        pgcd:=v1; {Le PGCD du (T[i] et T[j])*}
        writeln('PGCD entre T[' ,i, ' ]et T[' ,j, ' ] =',pgcd);
        {écraser le pgcdMax si le nouveau pgcd soit > pgcdMax*}
        if(pgcd>pgcdMax) then pgcdMax:=pgcd;
      end;
  writeln('Le plus grand PGCD de ce tableau est: ',pgcdMax);

```

Pour chaque paire(T[i], T[j]) des 9 éléments de ce tableau

```
readkey;  
end.
```

I.11. Exercice 9 (Fusionner deux tableaux)

Étant donné deux tableaux d'entiers **T1** et **T2** de 10 cellules chacun, écrire un programme qui permet de fusionner les éléments des deux tableaux **T1** et **T2** dans un tableau **T3** de 20 cellules en respectant l'ordre croissant dans ce dernier, et finalement afficher le tableau **T3**.

Solution

```
program Exercice1_9;  
uses crt;  
const n=10;  
      m=2*n;  
type  
  Tab1=array[1..n]of integer;  
  Tab2=array[1..m]of integer;  
var T1,T2:Tab1; T3:Tab2; i,j,k,v:integer;  
begin  
  writeln('la saisie du 1er Tableau ');  
  for i:=1 to n do  
    begin  
      write('Entrer La valeur N: ',i,': ');  
      readln(T1[i]);  
    end;  
  writeln('la saisie du 2eme Tableau ');  
  for i:=1 to n do  
    begin  
      write('Entrer La valeur N: ',i,': ');  
      readln(T2[i]);  
    end;  
  {*Trier le 1er tableau*}  
  for i:=1 to n-1 do  
    for j:=i+1 to n do
```



```

begin
    if (T1[j]<T1[i]) then
        begin
            v:=T1[i];
            T1[i]:=T1[j];
            T1[j]:=v;
        end;
    end;
for i:=1 to n-1 do  {*Trier le 2eme tableau*}
    for j:=i+1 to n do
        begin
            if (T2[j]<T2[i]) then
                begin
                    v:=T2[i];
                    T2[i]:=T2[j];
                    T2[j]:=v;
                end;
            end;
i:=1;
j:=1;
for k:=1 to m do {*trouver les 20 éléments du 3ème tableau T3*}
    begin
        if (i<=n) and (j<=n) then
            if (T1[i]<T2[j]) then
                begin
                    T3[k]:=T1[i];
                    i:=i+1;
                end
            else
                begin
                    T3[k]:=T2[j];
                    j:=j+1;
                end
            end
        end
    end

```

La k^{ème} valeur du tableau T3.
T3[k] := T1[i]; si T1[i]<T2[j]

La k^{ème} valeur du tableau T3.
T3[k] := T2[j]; si T2[j]≤T1[i]

```

else
    if (i>n) then
        T3[k] := T2[j]
    else
        T3[k] := T1[i];
    end;
for k:=1 to m do {*afficher les 20 éléments du tableau T3*}
    writeln('la valeur N ', k, '= ', T3[k]);
readkey;
end.

```

T3[k]:=T2[j] si toutes les valeurs du 1^{er} Tableau sont insérées dans le tableau T3 ; et T3[k]:=T1[i] sinon.

I.12. Rappel (Les tableaux bidimensionnels)

Une matrice (Tableau bidimensionnel) **M** de dimension $n \times k$ est un tableau de dimension n dont chaque élément est un tableau de dimension k .

Exemple : En Pascal nous pouvons déclarer une matrice **M** d'éléments de type entier et qui comporte 2 lignes et 3 colonnes de la manière suivante :

```

Const n=2 ;
      k=3 ;
Type Matrice= array [1..n, 1..k] of integer;
Var M :Matrice ;

```

Nombre de lignes

Nombre de colonnes

Dans ce cas nous pouvons stocker 6 éléments.

Chaque élément de la matrice est accessible via ses coordonnées (indices) i, j qui représentent respectivement le numéro de ligne et le numéro de colonne de cet élément.

32	53	81
68	17	21

Par exemple, nous pouvons récupérer l'élément à l'indice 6 : $M[2,3]=21$.

I.13. Questions

Comment pouvez-vous demander à l'utilisateur d'entrer les éléments d'une matrice de 15 lignes et 10 colonnes de type caractère ?

Comment pouvez-vous afficher les éléments de cette matrice ?

Réponses

```

Program Questions2;
Uses crt;
Const M=15;
        N=10;
Type Matrice=array[1..M,1..N]of char;
var M:Matrice;
        i,j:integer;
begin
    for i:=1 to M do
        for j:=1 to N do
            begin
                write('Entrer la valeur de A[' ,i ,', ',j ,']:');
                read(A[i,j]);
            end;
    for i:=1 to M do
        begin
            for j:=1 to N do
                write(C[i,j], ' ');
            end;
            writeln;
        end;
    readkey ;
end.

```

L'insertion les éléments de la matrice

Affichage les éléments de la matrice

I.14. Exercice 10 (Matrice binaire)

Étant donné une matrice binaire **M** de taille (20×12) initialisée aléatoirement à travers la fonction `random(2)` qui permet de générer une valeur inférieure à 2.

Sachant que chaque ligne de la matrice représente un nombre binaire codé sur 12 bits, enregistrer les valeurs équivalentes en décimal dans un tableau **T**.

Solution

```

program Exercice1_10;
uses crt,math;

```

```

const n=20;
        m=12;
type Matrice=array[1..n,1..m]of integer;
        tab=array[1..n]of integer;
var Mat:Matrice;
        T:Tab;
        i,j:integer; v:real;
begin
    Randomize;
    for i:=1 to n do {*pour chaque ligne i de la matrice Mat*}
        for j:=1 to m do {*pour chaque collone j de la ligne i *}
            Mat[i,j]:=random(2);
        for i:=1 to n do
            begin {*afficher les valeurs de la matrice Mat *}
                write('La valeur N',i:2,' en binaire=');
                for j:=1 to m do write(Mat[i,j],' ');
                writeln;
            end;
            {*calculer la valeur T[i] a partir de la ligne i de la Matrice Mat*}
            for i:=1 to n do
                begin
                    v:=0;
                    for j:=m downto 1 do
                        v:=v+Mat[i,j]*power(2,m-j);
                    T[i]:=round(v); cal
                end;
            for i:=1 to n do {*afficher les valeur decimale calculées*}
                writeln('La valeur N',i:2,' en decimal= ',T[i]);
            readkey;
        end.

```

Mat[i,j] prends une valeur aléatoire comprise entre 0 et 1

Calculer la valeur equivalente à la valeur binaire enregistrée dans la ligne i de la matrice Mat

Sachant que la fonction `power (base,exponent: float):float;` retourne A^B .

La fonction `Round (v:Real):Integer;` retourne la valeur arrondie d'un nombre réel v au nombre entier le plus proche.

I.15. Exercice 11 (modifier les éléments de la matrice)

Étant donné une matrice M d'entiers compris entre 1 et 20 initialisée aléatoirement de taille (15×20), changer tous les éléments de la ligne « i » et de la colonne « j » en 10 si la valeur de la cellule $M[i, j]=10$. Le programme doit afficher la matrice M avant et après la modification.

Solution

```

program Exercice1_11;
  uses crt;
  const n=15; p=20;
  type Matrice=array[1..n,1..p]of integer;
  var M:Matrice; i,j,k:integer;
begin
  Randomize;
  {*insérer les éléments des 15 lignes et 20 collones de la matrice M*
  for i:=1 to n do
    for j:=1 to p do M[i,j]:=1+random(20);
  writeln('les elements de la matrice avant le changement');
  for i:=1 to n do
    begin
      Affichage des éléments de la matrice M
      for j:=1 to p do write(M[i,j]:2, ' ');
      writeln;
    end;
  for i:=1 to n do
    for j:=1 to p do
      Pour chaque valeur M[i,j] de ligne i et de la colonne j de la matrice M,
      if (M[i,j]=10) then {*si M[i,j]=10*
        begin
          {*marquer toutes les cases de la ligne i par -1*
          for k:=1 to p do if (M[i,k]<>10) then M[i,k]:=-1;
          {*marquer toutes les cases de la colonne j par -1*
          for k:=1 to n do if (M[k,j]<>10) then M[k,j]:=-1;
        end;
  {*Chaque valeur T[i,j] de la matrice marquée par -1 prends la valeur 10*
  for i:=1 to n do
    for j:=1 to p do if (M[i,j]==-1) then M[i,j]:=10;

```

```
writeln('les elements de la matrice apres le changement');
for i:=1 to n do
  begin
    for j:=1 to p do
      write(M[i,j]:2, ' ');
    writeln;
  end;
readkey;
end.
```

I.16. Exercice 12 (La ligne ayant la plus grand somme)

Ecrire un programme Pascal qui demande à l'utilisateur de remplir une matrice d'entiers **M** de taille (4×5) et qui permet de trouver et d'afficher la ligne ayant la plus grande somme.

Solution

```
program Exercice1_12;
uses crt;
const n=4;
      k=5;
type
  Matrice=array[1..n,1..k]of integer;
var
  M:Matrice;
  i,j,sommeMax,sommeline,ligne:integer;
begin
  (*insérer les éléments des 4 lignes et 5 collones de la matrice M*)
  for i:=1 to n do
    for j:=1 to k do readln(M[i,j]);
  (*calculer la somme des 5 colonnes de la 1ère ligne de la matrice M*)
  sommeline:=0;
  for j:=1 to k do sommeline:=sommeline+M[1,j];
  sommeMax:=sommeline;(*initialiser sommeMax par la somme calculée*)
  ligne:=1; (*initialiser la ligne ayant sommeMax par 1)
```

```

for i:=2 to n do {*pour chaque ligne de 2 à n*}
    begin
    {*calculer la somme des 5 colonnes de la ième ligne de la matrice M*}
        sommeline:=0;
        for j:=1 to k do sommeline:=sommeline+M[i,j];
        if(sommeline>sommeMax) then
            begin
                sommeMax:=sommeline;
                ligne:=i;
            end;
        end;
        writeln('la ligne ',ligne,' a la plus grande somme ='
            ,sommeMax);
        readkey;
    end.

```

Ecraser sommeMax si la somme des valeurs de la ligne i sommeline>sommeMax

I.17. Exercice 13 (les lignes et colonnes dont la somme est supérieure ou égale à une valeur)

Étant donné une matrice **M** de taille (12×10) d'entiers compris entre 0 et 99 initialisée aléatoirement à travers la fonction `random(100)`, afficher toutes les lignes et colonnes ayant une somme supérieure ou égale à une valeur **v** entrée par l'utilisateur.

Solution

```

program Exercice1_13;
    uses crt;
    const n=12;
           p=10;
    type
        Matrice=array[1..n,1..p]of integer;
    var
        M:Matrice;
        i,j,v,s:integer;
    begin
        Randomize;

```

```

for i:=1 to n do
  for j:=1 to p do
    M[i,j]:=random(100);
  for i:=1 to n do {*Afficher les éléments de la matrice M*}
    begin
      for j:=1 to p do write(M[i,j]:4);
      writeln;
    end;
write('Entrer la valeur v SVP: ');
read(v);
for i:=1 to n do {*pour chaque lignes i de la matrice M*}
  begin
    s:=0; {*calculer la somme s de la ligne i de la matrice M*}
    for j:=1 to p do s:=s+M[i,j];
    if(s>=v) then {*afficher les éléments de la ligne i si s>=v*}
      begin
        write('ligne N',i,': ');
        for j:=1 to p do write(M[i,j] :4);
        writeln(' La somme =',s);
      end;
  end;
for j:=1 to p do {*pour chaque colonne j de la matrice M*}
  begin
    s:=0;{*calculer la somme s de la colonne j de la matrice M*}
    for i:=1 to n do s:=s+M[i,j];
    if(s>=v) then {afficher les éléments de la colonne j si s>=v}
      begin
        write('colonne N',j,': ');
        for i:=1 to n do write(M[i,j] :4);
        writeln(' La somme =',s);
      end;
  end;
end;
readkey;
end.

```

Générer une valeur aléatoire comprise entre 0 et 99 pour chaque élément de la matrice M

I.18. Exercice 14 (Chaque élément est remplacé par le plus petit l'élément à sa droite)

Étant donné une matrice de taille (7×10) d'entiers compris entre 50 et 90 initialisée aléatoirement, écrire un programme efficace pour remplacer chaque élément de la matrice par le plus petit élément à sa droite s'il est inférieur à celui-ci. Le programme doit afficher la matrice avant et après le changement.

Solution

```

program Exercice1_14;
  uses crt;
  const n=7; m=10;
  type Matrice=array[1..n,1..m]of integer;
  var T:Matrice;
      i,j,v,k:integer;
begin
  Randomize;
  {*initialiser les éléments de la matrice*}
  for i:=1 to n do
    for j:=1 to m do T[i,j]:=50+random(40);
  {*afficher les éléments de la matrice*}
  writeln('la matrice avant le changement');
  for i:=1 to n do
    begin
      for j:=1 to m do write(T[i,j],#9);
      writeln;
    end;
  for i:=1 to n do
    for j:=1 to m do
      begin
        v:=T[i,j];
        for k:=j+1 to m do
          if (T[i,k]<v) then v:=T[i,k];
        T[i,j]:=v; {remplacer T[i,j]par la plus petite valeur trouvée}
      end;

```

Chercher la plus petite valeur v à droite de l'élément de la colonne j

```

{*afficher les éléments de la matrice*}
writeln('la matrice apres le changement');
for i:=1 to n do
    begin
        for j:=1 to m do
            write(T[i,j] :4);
        writeln;
    end;
readkey;
end.

```

I.19. Exercice 15 (Les éléments de pic)

Étant donné une matrice carrée M de taille (15×15) de réels compris entre 0 et 1 initialisée aléatoirement, écrire un programme Pascal qui affiche la matrice M et qui permet de trouver les indices (i et j) des éléments de pic.

Un élément $M[i, j]$ est considéré comme un pic si sa valeur v est supérieur strictement à toutes ses valeurs voisines (horizontales, verticales et diagonales).

Note : Il peut y avoir plus d'un élément de pic dans une matrice et la solution doit signaler tous les éléments de pic.

Solution :

```

program Exercice1_15;
uses crt;
const n=15;
type Matrice=array[1..n,1..n]of real;
var T:Matrice;
    i,j:integer;
begin
    {*intialiser les éléments de la matrice*}
    Randomize;
    for i:=1 to n do
        for j:=1 to n do
            T[i,j]:=random();

```

```

(*afficher les éléments de la matrice*)
for i:=1 to n do
  begin
    for j:=1 to n do
      write(T[i,j]:5:2);
      writeln;
    end;
  for i:=2 to n-1 do
    for j:=2 to n-1 do
      if (T[i,j]>T[i-1,j]) and (T[i,j]>T[i+1,j])
      and (T[i,j]>T[i,j-1]) and (T[i,j]>T[i,j+1])
      and (T[i,j]>T[i-1,j-1]) and (T[i,j]>T[i+1,j+1])
      and (T[i,j]>T[i-1,j+1]) and (T[i,j]>T[i+1,j-1]) then
        writeln(i,';',j);
      readkey;
    end.
  
```

Afficher les indices des éléments de la matrice qui se considèrent des éléments de pics

I.20. Exercice 16 (Les indices d'équilibres d'une matrice)

Étant donné une matrice **M** d'une taille (15×10) d'entiers compris entre 1 et 3 initialisée aléatoirement, écrire un programme qui affiche la matrice **M** et qui trouve et affiche l'indice d'équilibre *j* pour chaque ligne *i*.

L'indice *j* est considéré comme indice d'équilibre de la ligne *i* si la somme des éléments du sous-tableau gauche est égale à la somme des éléments du sous-tableau droit c'est-à-dire :

$$\sum_{k=1}^{j-1} M[i,k] = \sum_{k=j+1}^n M[i,k]$$

Le programme doit afficher "aucun indice d'équilibre" s'il n'existe pas un indice d'équilibre sur la ligne *i*.

Solution

```

program Exercicel_16;
uses crt,math;
const n=15;
      p=10;
type Matrice=array[1..n,1..p]of integer;
  
```

```

var M:Matrice;
    i,j,k,s,sLigne:integer;
begin
Randomize;
{*initialiser les éléments de la matrice*}
for i:=1 to n do
    for j:=1 to p do M[i,j]:=1+random(3);
{*afficher les éléments de la matrice*}
writeln('les elements de la matrice sont');
for i:=1 to n do
    begin
        for j:=1 to p do write(M[i,j]:4, ' ');
        writeln;
    end;
for i:=1 to n do
    begin
        sLigne:=0;
        for k:=1 to p do
            sLigne:=sLigne+M[i,k];
        j:=1;
        s:=0;
        while (sLigne-M[i,j]-2*s>0) do
            begin
                s:=s+M[i,j];
                j:=j+1;
            end;
        if(sLigne-M[i,j]-2*s=0) then
            writeln('La ligne N ',i:2,' :L''indice
                    d''equilibre: ',j)
        else writeln('La ligne N ',i:2,' :aucun indice
                    d''equilibre');

    end;
readkey;
end.

```

Calculer la somme des p colonnes de la i^{ème} ligne (sLigne)

Chercher la colonne j qui divise la i^{ème} ligne en 2 sous tableaux quasi-equivalents

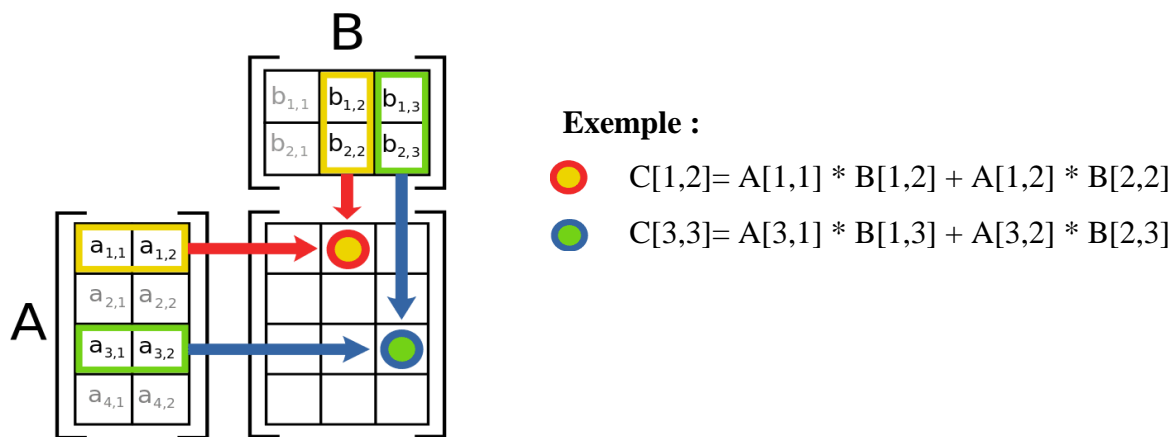
Afficher l'indice j s'il est considéré comme indice d'équilibre de la i^{ème} ligne

I.21. Exercice 17 (Multiplier deux tableaux bidimensionnels)

La multiplication d'une matrice par une autre matrice est une opération mathématique courante. Les développeurs utilisent généralement l'algorithme de multiplication matricielle pour cette opération.

Fonctionnement de la multiplication matricielle :

Soit **A** une matrice avec *m* lignes et *p* colonnes. De même, **B** est une matrice avec *p* lignes et *n* colonnes. Multipliez la matrice **A** par la matrice **B** pour produire une matrice **C**, avec *m* lignes et *n* colonnes. Chaque élément $C[i, j]$ de la matrice **C** est obtenue en multipliant toutes les éléments de la *i*^{ème} ligne de la matrice **A** par les éléments correspondants dans la *j*^{ème} colonne de la matrice **B**, puis en ajoutant les résultats. La figure suivante illustre ces opérations.



Ecrire un programme en Pascal qui demande à l'utilisateur de saisir les valeurs d'une matrice **A** de taille 4×2 et les valeurs d'une matrice **B** de taille 2×3, et qui permet de calculer et d'afficher la matrice **C** représentant la multiplication de **A** par **B**.

Solution

```

program Exercice1_17;
uses crt;
Const M=4;P=2;N=3;
Type MatA=array[1..M,1..P]of real;
      MatB=array[1..P,1..N]of real;
      MatC=array[1..M,1..N]of real;
var A:MatA;
      B:MatB;
      C:MatC;
    
```

```

        i,j,k:integer;
begin
    {Insertion des éléments de la matrice A}
    for i:=1 to M do
        for j:=1 to P do
            begin
                write('Entrer la valeur de A[' ,i ,', ' ,j ,']:');
                read(A[i,j]);
            end;
        {Insertion des éléments de la matrice B}
        for i:=1 to P do
            for j:=1 to N do
                begin
                    write('Entrer la valeur de B[' ,i ,', ' ,j ,']:');
                    read(B[i,j]);
                end;
            {Initialiser la matrice C avec des zeros}
            for i:=1 to M do
                for j:=1 to N do
                    C[i, j] := 0;
                {calculer la matrice Calculer la matrice C}
                for i:=1 to M do
                    for j:=1 to N do
                        for k:=1 to P do
                            C[i, j] := C[i, j] + (A[i, k] * B[k, j]);
                        {Affichage de la matrice C}
                        for i:=1 to M do
                            begin
                                for j:=1 to N do write(C[i,j] :5:2);
                                    writeln;
                                end;
                            readkey;
                        end.
                    
```

I.22. Conclusion

Les tableaux unidimensionnels et bidimensionnels sont certainement les structures de données les plus populaires en programmation. L'intérêt de ces structures est de pouvoir stocker en mémoire une liste de variables de même type et dont chaque variable de données est accessible à l'aide d'un indice. Dans un tableau il est possible d'introduire des données de tous les types (boolean, integer, real, char, string.. etc) ; nous avons utilisé ces structures de données pour résoudre une multitude de problèmes dans ce chapitre.

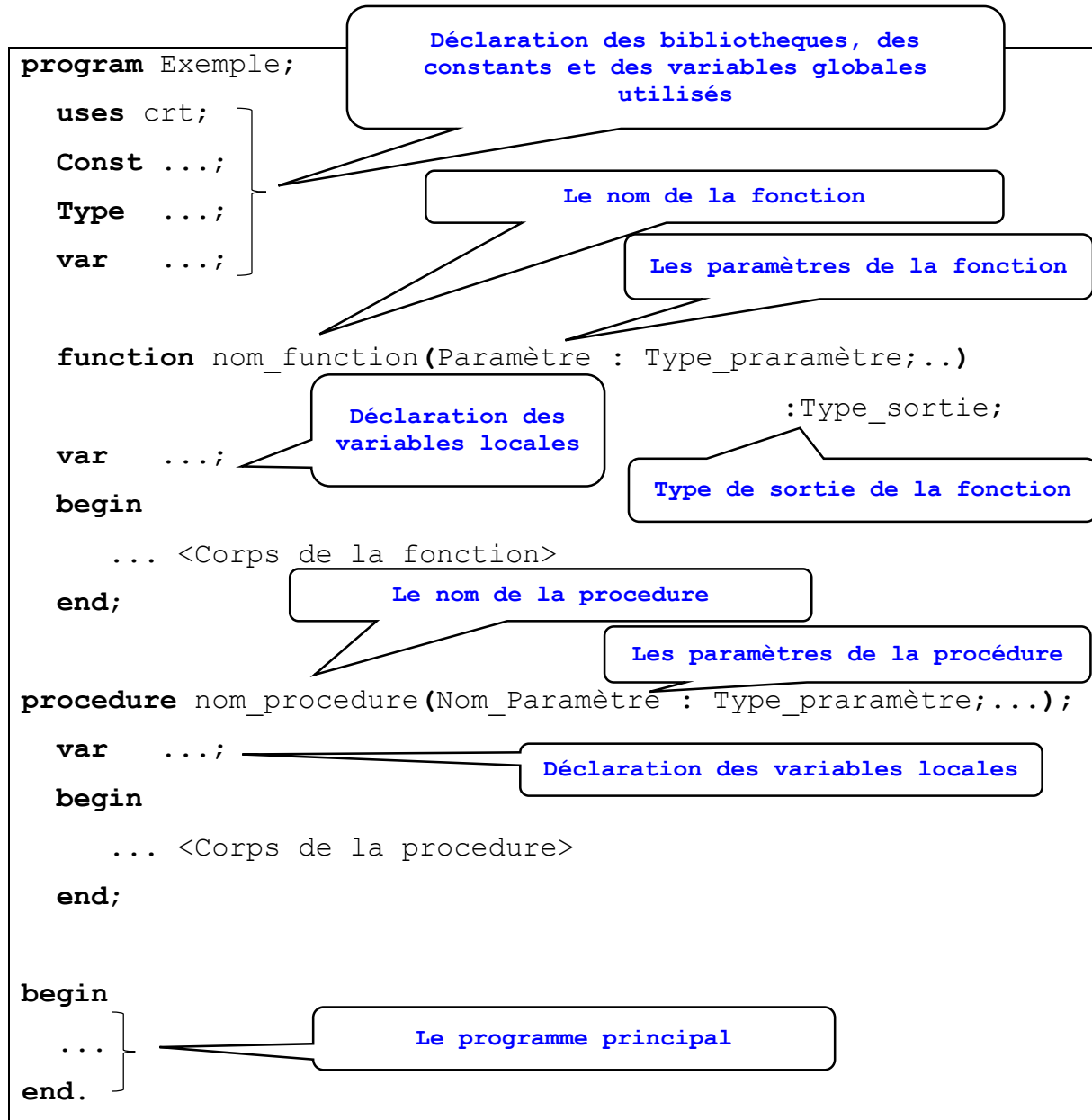
Chapitre

2

Les Procédures et fonctions

II.1. Les procédures et fonctions (Rappel)

En Pascal les sous-programmes (procédures ou fonctions) sont des blocs d'instructions nommées et déclarées dans l'entête du programme et appelées à chaque fois que le programmeur en a besoin.



Un sous-programme est un ensemble d'instructions qui prend en paramètre certaines entrées et effectue certaines tâches, l'appel d'un sous-programme s'effectue en spécifiant son nom et éventuellement ses paramètres ; cela déclenche l'exécution des instructions de ce sous-programme.

Un sous-programme utilise généralement les variables globales. Il peut aussi utiliser ses propres variables (dites locales) déclarées dans l'espace qui lui est réservé ; mais elles ne peuvent être accessibles que dans ce sous-programme.

Un sous-programme peut être appelé plusieurs fois en le paramétrant différemment à chaque appel. Une fonction a pour but principal de retourner une valeur, il est donc nécessaire de préciser le type de la fonction qui est en réalité le type de cette valeur.

Il existe deux façons de transmettre des paramètres aux procédures et aux fonctions :

- ✓ **Passage par valeur**
- ✓ **Passage par adresse**

Dans le cas du **passage par valeur**, une copie des paramètres réels est transmise aux arguments formels respectifs du sous-programme. Alors que, dans le cas du **passage par adresse**, l'adresse (emplacement dans la mémoire) des paramètres réels est transmise à des arguments formels du sous-programme, toute modification apportée aux arguments formels se reflétera également dans les arguments réels. En langage de programmation **Pascal** Le deuxième cas se distingue du premier cas par la présence du mot clé **var** préfixant la définition du paramètre formel dans l'entête du sous-programme

Exemple

```
program exchange;
  var x,y : integer;
  procedure swap_par_val(a,b : integer);
  var t : integer;
  begin
    t := a ; a := b ; b := t;
  end;

  procedure swap_par_adr(var a,b : integer);
  var t : integer;
  begin
    t := a ; a := b ; b := t;
  end;

  begin
    readln (x,y);
    swap_par_val(x,y);
    writeln('x= ',x,' y=',y) ;
    swap_par_adr(x,y);
    writeln('x= ',x,' y=',y) ;
  end.
```

Dans le passage par valeur, lors de l'appel procédurale `swap_par_val(x,y)`, les valeurs des paramètres **x** et **y** sont copiées à un autre emplacement dans la mémoire. Lors de l'accès ou de la modification des variables **a** et **b** dans la procédure `swap_par_val(a,b:integer)`, on accède uniquement aux copies **a** et **b**. Ainsi, il n'y a aucun effet sur les valeurs de **x** et **y** ; dans ce cas le programme affiche les valeurs originales de **x** et **y**.

Dans le cas du passage par adresse, lors de l'appel procédurale `swap_par_adr(x,y)`, les adresses mémoires des variables `x` et `y` sont transmises à la procédure `swap_par_adr(var a,b:integer)`. En d'autres termes, cette procédure accède directement aux variables réelles `x` et `y`, dans ce cas cette procédure permet d'échanger le contenu des variables `x` et `y`. Puisqu'on veut modifier le contenu des variables `x` et `y`, il faut faire un passage de paramètres par adresse. D'où la présence du mot clé `var` dans la définition des paramètres `a` et `b` de l'entête.

II.2. Exercice 1 (Plus proche de 20)

Ecrivez la fonction `plusProcheDe20(n,m : integer) :integer` ; qui retourne le nombre le plus proche de la valeur 20 parmi les deux entiers `n` et `m`. cette fonction retourne 0 si l'écart de `n` et `m` avec la valeur 20 est le même.

Solution

```

program Exercice2_1;
Uses crt;
var a,b:integer;
    function plusProcheDe20(n,m : integer) :integer ;
    begin
    {*la fonction retourne 0 si l'écart entre 20 et n = l'écart entre 20 et m
    et retourne n si l'écart entre 20 et n < l'écart entre 20 et m et
    retourne m sinon*}
        if (abs(n-20)=abs(m-20)) then plusProcheDe20:=0
        else if (abs(n-20)<abs(m-20)) then plusProcheDe20:=n
            else plusProcheDe20:=m;
    end;
begin
    write('Entrer la valeur de a SVP: ');
    readln(a);
    write('Entrer la valeur de b SVP: ');
    readln(b);
    writeln(plusProcheDe20(a,b));
    readkey ;
end.

```

Afficher le resultat retournée par la fonction plusProcheDe20 avec les paramètres a et b entrées par l'utilisateur

Sachant que la fonction `abs(n)` retourne la valeur absolue de `n`.

II.3. Exercice 02 (somme / double)

Ecrivez en Pascal la fonction **somme(n,m :integer) :integer** ; qui retourne la somme des deux valeurs **n** et **m**. Si les deux valeurs sont identiques, cette fonction retourne le double de leurs somme.

Solution

```

program Exercice2_2;
Uses crt;
var a,b:integer;

    function somme(n,m:integer):integer;
    var res:integer;
    begin
        res:=n+m; { * res est la somme n et m *}
        { *si n=m res devient le double de la somme n et m*}
        if (n=m) then
            res:=res*2;
        somme:=res;
    end;

begin
    write('Entrer la valeur de a SVP: ');
    readln(a);
    write('Entrer la valeur de b SVP: ');
    readln(b);
    writeln(somme(a,b));
    readkey ;
end.

```

Afficher le resultat retournée par la fonction somme avec les paramètres a et b entrées par l'utilisateur

II.4. Exercice 3 (Approximation d'une expression)

Ecrire la fonction **approximation_E(n :integer) :real** ; qui permet de calculer et retourner la valeur de l'expression E suivante :

$$E = \frac{\sum_{i=1}^n i}{n!}$$

Ecrire un programme en Pascal qui demande à l'utilisateur de saisir un entier strictement positif **A** et qui permet d'afficher l'approximation de la fonction **E** jusqu'au rang **A** :

Solution

```
program Exercice2_3;
Uses crt;
var a:integer;
    approx:real;

    function approximation_E(n:integer):real;
var i:integer; fact,somme:longint;
begin
    somme:=0;
    for i:=1 to n do
        somme:=somme+i;
    fact:=1;
    for i:=1 to n do
        fact:=fact*i;
    (* Le resultat retourné par la fonction approximation_E *)
    approximation_E:=somme/fact;
end;

begin
    repeat
        write('Entrer la valeur de a SVP: ');
        readln(a);
        if(a<=0) then writeln('erreur de saisie') ;
    until a>0;
    (* approx reçoit le resultat retournée par la fonction approximation_E
    En passant en paramètre la valeur 'a' entrée par l'utilisateur*)
    approx:=approximation_E(a);
    (*Afficher L'approximation de E jusqu'au rang 'a'*)
    writeln('approximation de jusqu au rang ',a,'=',approx);
    Readkey ;
end.
```

Calculer la somme des n premier entiers

Calculer la la factorielle de n

II.5. Exercice 4 (Les nombres premiers)

Donner le code de :

- La fonction **premier(v :integer) :boolean** ; qui retourne **True** si la valeur **v** est premier (v est considérée premier s'il admet seulement 2 diviseurs : 1 et v).
- La procédure **premierSuivant(var v :integer)**; qui cherche la première valeur premier juste après la valeur **v** en se basant sur le passage par adresse.
- La procédure **remplirTableau(val :integer)** ; qui permet de remplir le tableau **T** par les **15** premiers nombres premiers supérieurs à **val**.
- La procédure **afficher(T :Tab)** ; qui affiche les éléments du tableau.

Ecrire le programme qui demande à l'utilisateur d'entrer une valeur strictement positif **a** et qui permet remplir le tableau **T** à travers l'appel procédural **remplirTableau(a)**, puis afficher les éléments de ce Tableau.

Solution

```

Program Exercice2_4 ;
uses crt;
const n=15;
type Tab=array[1..n]of integer;
var T:Tab; a:integer;

function premier(v:integer):boolean;
  var i:integer;
begin
  i:=2;
  while(v mod i<>0)do i:=i+1;
  if(i=v)then Premier:=true
  else premier:=false;
end;

procedure premierSuivant(var v :integer);
begin
  v :=v+1 ;
  while(not premier(v))do v:=v+1;
end;

```

Un nombre 'v' est considéré premier si son premier diviseur >1 = 'v' lui meme

Chercher et retourner le 1^{er} entier premier supérieur strictement à 'v'

```
procedure remplirTableau(val:integer);
```

```
var i:integer;
```

```
begin
```

```
    for i:=1 to n do
```

```
        begin
```

```
            premierSuivant(val);
```

```
            T[i]:=val;
```

```
        end;
```

```
end;
```

```
procedure afficherTableau();
```

```
var i:integer;
```

```
begin
```

```
    for i:=1 to n do
```

```
        writeln('Le nombre premier N ',i,' = ',T[i]);
```

```
end;
```

```
begin
```

```
{*forcer l'utilisateur à saisir une valeur 'a' strictement positif*}
```

```
repeat
```

```
    write('Entrer la valeur de A SVP: ');
```

```
    readln(a);
```

```
    if (a<=0) then
```

```
        write('erreur! valeur négatif..');
```

```
    until (a>0);
```

```
    remplirTableau(a); {*Procédure qui permet de remplir le tableau*}
```

```
    afficherTableau(); {*Procédure qui permet d'afficher le tableau*}
```

```
    readkey;
```

```
end.
```

Chaque élément du tableau T reçoit le 1^{er} entier premier supérieur strictement à 'val'

Afficher le tableau T

II.6. Exercice 5 (Chiffre commun)

Ecrire la fonction **existeDans(c,n :integer) :boolean** ; qui permet de retourner vraie si le chiffre **c** existe dans le nombre **n**. Exemple : **existeDans(2,32)** retourne **vraie** par contre **existeDans(2,35)** retourne **faux**.

Donner le code de la fonction **chiffreCommun(n,k,m :integer) :boolean** ; sachant que n, k et m trois entiers positifs, chacun compris entre 0 et 99. Elle retourne **vraie** si un chiffre apparaît dans les trois nombres. **Exemple** : le 5 apparait dans 25, 55 et 15.

Solution

```

program Exercice2_5;
Uses crt;
var a,b,c:integer;

    function existeDans(c,n :integer) :boolean ;
        var u:integer;
            res:boolean;

    begin
        res:=false;
        while (n<>0) do
            begin
                u:=n mod 10;
                n:=n div 10;
                if (u=c) then res:=true;
            end;
        existeDans:=res;
    end;

    function chiffreCommun(n,k,m :integer) :boolean ;
    var u:integer; res:boolean;
    begin
        res:=false;
        while (n<>0) do
            begin
                u:=n mod 10;
                n:=n div 10;
                if (existeDans(u,k) and existeDans(u,m)) then
                    res:=true;
            end;
        chiffreCommun:=res;
    end;

```

La fonction existeDans retourne vraie s'il existe un chiffre 'u' retiré du nombre 'n' = chiffre 'c'

La fonction chiffreCommun retourne vraie s'il existe un chiffre 'u' retiré du nombre 'n', existe dans les deux chiffres 'k' et 'm'


```

begin
  (*forcer l'utilisateur à saisir une valeur 'a' appartient [0,99]*)
  repeat
    write('Entrer la valeur de A SVP: ');
    readln(a);
    if (a<0) or (a>=100) then writeln('Erreur de saisie. ');
  until (a>=0) and (a<=99);
  (*forcer l'utilisateur à saisir une valeur 'b' appartient [0,99]*)
  repeat
    write('Entrer la valeur de B SVP: ');
    readln(b);
    if (b<0) or (b>=100) then writeln('Erreur de saisie. ');
  until (b>=0) and (b<=99);
  (*forcer l'utilisateur à saisir une valeur 'c' appartient [0,99]*)
  repeat
    write('Entrer la valeur de C SVP: ');
    readln(c);
    if (c<0) or (c>=100) then writeln('Erreur de saisie. ');
  until (c>=0) and (c<=99);
  (*afficher le resultat retourné par la fonction chiffreCommun(a,b,c)*)
  writeln(chiffreCommun(a,b,c));
  readkey ;
end.

```

II.7. Exercice 6 (Les nombres parfaits)

Le nombre parfait c'est le nombre qui est égal à la somme de ses diviseurs. **Exemple :** l'entier 6 est un nombre parfait car $6 = 1 + 2 + 3$.

Donner le code de :

- La fonction **sommeDiviseurs(n :integer) :integer** ; qui calcule et retourne la somme des diviseurs de **n**.
- La fonction **parfait(n :integer) :boolean** ; qui retourne vraie si **n** est un nombre parfait, et retourne faux sinon.
- La procédure **parfaitsJusqua(n :integer)** ; qui permet d'afficher les nombres parfaits inférieurs ou égale à **n**.

Ecrire le programme qui demande à l'utilisateur un nombre entier positif **m** et qui affiche les nombres parfaits inférieurs ou égale à **m**.

Solution

```

Program Exercice2_6;

uses crt;
var m:integer;
function sommeDiviseurs(n :integer) :integer ;
var s,i:integer;
begin
    s:=0;
    for i:=1 to (n div 2) do
        if (n mod i=0) then s:=s+i;
    sommeDiviseurs:=s; {* 's' est la somme des diviseurs de 'n'*}
end;
function parfait(n :integer) :boolean ;
begin
    {*si la somme des diviseurs de 'n'=n ce nombre est considéré parfait*}
    if (sommeDiviseurs(n)=n) then parfait:=true
    else parfait:=false;
end;
procedure parfaitsJusqua(n:integer);
var i:integer;
begin
    for i:=1 to n do
        if (parfait(i)) then writeln(i, ' est parfait');
end;
begin
    write('Entrer la valeur la valeur de m SVP :');
    read(m);
    if (m>=6) then parfaitsJusqua(m)
    else write('aucun nombre parfait inferieur ', m);
    readkey;
end.

```

Un entier 'i' < (n div 2) est considéré diviseur de 'n' si 'n mod i=0'

Pour chaque valeur 'i' ≤ n ; Afficher 'i' s'il est parfait.

II.8. Exercice 7 (Les multiples)

Donner le code de la procédure **multiplesDe(n,v: entier)** ; qui permet d'afficher les multiples de **n** qui sont inférieur ou égale à la valeur **v**.

Ecrire un programme qui demande à l'utilisateur de saisir deux valeurs **a** et **b**, et qui permet d'afficher les multiples de **a** qui sont inférieur ou égale à la valeur **b** en utilisant la procédure précédente. Le programme doit prendre en compte tous les cas possibles y compris le cas d'erreur de saisi.

Solution

```
program Exercice2_7;
Uses crt;
var a,b:integer;
    procedure multiplesDe(n,v:integer);
    var m:integer;
    begin
        if(n>v) then writeln('erreur de saisi ',n,'>',v)
        else
            begin (* Afficher les multiples de 'n' < 'v' *)
                writeln('les multiples de ',n,' < ',v,' sont: ');
                while(m<=v) do
                    begin
                        writeln('    ',m);
                        m:=m+n;
                    end;
                end;
            end;
    end;
begin
    write('Entrer la valeur de a SVP: ');
    readln(a);
    write('Entrer la valeur de b SVP: ');
    readln(b);
    multiplesDe(a,b);
    readkey ;
end.
```

Demander à l'utilisateur deux valeurs 'a' et 'b', puis afficher les multiples de 'a' < 'b'

II.9. Exercice 8 (Les voisins croissants d'un tableau)

Ecrivez le code de la procédure **remplirTableau()** ; qui permet de remplir un tableau de **n** éléments entiers (**n=11**).

Ecrivez le code de la fonction **maxCroissants(T :Tab) :integer** ; qui permet de renvoyer le nombre maximum d'éléments voisins croissants d'un tableau **T**.

Un tableau est dite « trié » si le nombre maximum de ses éléments voisins croissants est égal à la taille du Tableau. Donner le code de la fonction **estTrie(T :Tab) :boolean** ; qui retourne **True** si le tableau est trié.

Solution

```
Program Exercice2_8 ;
uses crt;
const n=8;
type Tab=array[1..n]of integer;
var T:Tab;

procedure remplirTableau();
var
    i:integer;
begin
    (* saisir les 8 éléments du Tableau T*)
    for i:=1 to n do
        begin
            write('Entrer la valeur N ',i,': ');
            readln(T[i]);
        end;
    end;

function maxCroissants(T :Tab) :integer ;
var
    nb, nbmax, i: integer;
begin
    nb:=1;
    nbmax:=1;{le nombre max des éléments croissants initialisé à 1}
```

```

    for i:=2 to n do
        begin
            if (T[i]>T[i-1]) then
                begin
                    nb:=nb+1;
                    if (nb>nbmax) then
                        nbmax:=nb;
                    end
                end
            else nb:=1;
            end;
            maxCroissants:= nbmax;
        end;
    function estTrie(T:Tab):boolean;
    begin
        if (maxAdjacents (T)=n) then
            estTrie:=True
        else
            estTrie:=false;
        end;
    begin
        remplirTableau();
        if (estTrie(T)) then
            write('Tableau Trié')
        else
            write('Tableau n'est pas trié');
        readkey;
    end.

```

Ecraser la valeur 'nbmax', s'il existe nombre max des éléments croissants 'nb' > 'nbmax'

Un tableau est considéré trié si le max des éléments croissants = 'n'

II.10. Exercice 9 (Calcul matriciel)

- Donner le code de la fonction **nbPairs(M :Matrice) :integer** ; qui permet de compter le nombre d'éléments pairs d'un Matrice **M**.
- Donner le code de la fonction **nbImpairs(M :Matrice) :integer** ; qui permet de compter le nombre d'éléments impairs d'un Matrice **M**.

- c. Donner le code de la fonction **moyenne(M :Matrice) :real** ; qui permet de calculer la moyenne d'une Matrice **M**.
- d. Donner le code de la procédure **statistiques (M : Matrice)** ; qui permet d'afficher la moyenne, le nombre d'éléments pairs et impairs de la matrice **M**.
- e. Ecrire un programme Pascal qui demande à l'utilisateur de remplir une matrice de taille (7×4) et qui permet d'afficher les statistiques de cette matrice à travers la procédure **statistiques (M : Matrice)**.

Solution

```

program Exercice2_9;
uses crt;
const n=7;k=4;
type Matrice=array[1..n,1..k]of integer;
var M1:Matrice;

function moyenne(M:Matrice):real;
var i,j, somme:integer;
begin
    somme:=0;
    for i:=1 to n do
        for j:=1 to k do
            somme:=somme+M[i,j];
    moyenne:=somme/(n*k);{*calculer la moyenne de la matrice M*}
end;

function nbPairs(M:Matrice):integer;
var i,j, nb:integer;
begin
    nb:=0;
    for i:=1 to n do
        for j:=1 to k do
            if(M[i,j] mod 2=0) then nb:=nb+1;
    nbPairs:=nb;
end;

```

Calculer la somme des éléments de la matrice M

Compter le nombre des éléments paires 'nb'. Un élément M[i,j] est considéré paire s'il est divisible par 2

```

function nbImpairs (M:Matrice):integer;
var i,j, nb:integer;
begin
    nb:=0;
    for i:=1 to n do
        for j:=1 to k do
            if (M[i,j] mod 2=1) then
                nb:=nb+1;
    nbImpairs:=nb;
end;

procedure statistiques (M:Matrice);
begin
    writeln('La moyenne des elements = ', moyenne(M));
    writeln('Nombre d'elements pairs= ', nbPairs(M)) ;
    writeln('Nombre d'elements impairs= ', nbImpairs(M)) ;
end;

begin
    (* saisir les éléments de la matrice M1*)
    for i:=1 to n do
        for j:=1 to k do
            readln(M1[i,j]);
    statistiques(M1); (* afficher les statistiques de la matrice M1*)
    readkey;
end.

```

Compter le nombre des éléments impaires 'nb'.
Un élément M[i,j] est considéré impaire s'il
n'est pas divisible par 2

II.11. Exercice 10 (Matrice équilibrée)

- a. Donner le code de la procédure **remplirMatrice()** ; qui permet de remplir une matrice carrée de taille (4×4).
- b. Donner le code de la fonction **sommeLigne(l :integer ; M :Matrice) :integer** ; qui permet de calculer et retourner la sommes des éléments de la ligne N° *l* d'une matrice *M* de taille (4×4).

- c. Donner le code de la fonction **sommeColonne(c :integer ; M :Matrice) :integer ;** qui permet de calculer et retourner la sommes des éléments de la colonne N° c d'une matrice **M** de taille (4×4).
- d. Donner le code de la fonction **equilibree (M : Matrice) :boolean ;** qui permet de vérifier si la matrice **M** est équilibrée, sachant qu'une matrice équilibrée est une matrice où la somme des éléments de chaque ligne **i** de la matrice **M** est égale à la somme des éléments de la colonne **i**.
- e. Ecrire un programme Pascal qui demande à l'utilisateur de remplir une matrice à travers la procédure **remplirMatrice()** ; et qui permet de vérifier l'équilibrage de cette matrice.

Solution

```

program Exercice2_10;
uses crt;
const n=4;
type Matrice=array[1..n,1..n]of integer;
var M1:Matrice;

procedure remplirMatrice();
var
    i,j:integer;
begin
    (* saisir les éléments de la matrice M1*)
    for i:=1 to n do
        for j:=1 to n do
            readln(M1[i,j]);
end;

function sommeLigne(l:integer; M:Matrice):integer;
var j,somme:integer;
begin
    somme:=0;
    for j:=1 to n do
        somme:=somme+M[l,j];
    sommeLigne:=somme;
end;

```

Calculer la somme des éléments de la ligne 'l' de la matrice M


```

function sommeColonne(c:integer; M:Matrice):integer;
var somme,i:integer;
begin
    somme:=0;
    for i:=1 to n do
        somme:=somme+M[i,c];
    sommeColonne:=somme;
end;

function equilibree(M:Matrice):boolean;
var res:boolean;
    i:integer;
begin
    /* La matrice M est initialisée équilibré */
    res:=true;
    i:=1;
    while res and (i<=n) do
        begin
            if (sommeLigne(i,M) <> sommeColonne(i,M)) then
                res:=false;
            i :=i+1 ;
        end;
    equilibree:=false;
end;

begin
    /*appeler la procedure remplirMatrice pour saisir les éléments de M1*/
    remplirMatrice();
    if (equilibree(M1)) then
        write('La matrice est equilibree')
    else
        write('La matrice n''est pas equilibree');
    readkey;
end.

```

Calculer la somme des éléments de la colonne 'c' de la matrice M

Une matrice devient inéquilibrée si la sommes des éléments de la ligne 'i' soit différent de la somme des éléments de la colonne 'i'

II.12. Exercice 11 (Tableau centré)

Ecrivez en Pascal le code des procédures et fonctions suivantes :

- La procédure **remplirTableau()** ; qui demande à l'utilisateur de remplir un tableau de 8 entiers positifs.
- La fonction **sommePairs(T :Tab) :integer** ; qui calcule et retourne la somme des éléments pairs d'un tableau d'entiers.
- La fonction **sommeImpairs(T :Tab) :integer** ; qui calcule et retourne la somme des éléments impairs d'un tableau d'entiers.
- Un Tableau est dite (**centré vers 10**) si l'écart entre la somme des éléments pairs et les éléments impairs de ce tableau est inférieur à 10. Donner le code de la procédure **centreVers10 (T :Tab)** ; qui affiche un message précisant si le tableau est centré vers 10 ou non.

Solution

```
Program Exercice2_11 ;
uses crt;
const n=8;
type Tab=array[1..n]of integer;
var T:Tab;

procedure remplirTableau();
var i:integer;
begin
    i:=1;
    (* forcer l'utilisateur à remplir le tableau T par des valeurs positifs*)
    while (i<=n) do
        begin
            write('Entrer la valeur N ',i,': ');
            readln(T[i]);
            if(T[i]>=0) then i:=i+1
            else
                writeln('Erreur!valeur négatif. ');
        end;
    end;
end;
```

```

function sommePairs(T :Tab) :integer ;
var i,s:integer;
begin
    s:=0;
    for i:=1 to n do
        if(T[i] mod 2 =0) then
            s:=s+T[i];
        sommePairs:=s;
    end;

function sommeImpairs(T :Tab) :integer ;
var i,s:integer;
begin
    s:=0;
    for i:=1 to n do
        if(T[i] mod 2<>0) then s:=s+T[i];
        sommeImpairs:=s;
    end;

procedure centreVers10 (T :Tab);
var ecart:real;
begin
    {calculer l'écart entre la somme des éléments pairs et impaires*}
    ecart:=abs(sommePairs(T)-sommeImpairs(T));
    if(ecart<10) then
        writeln('Tableau, centré vers 10')
    else
        writeln('Tableau, n''est pas centre vers 10');
    end;

begin
    remplirTableau();
    centreVers10(T);
    readkey;
end.

```

Calculer la somme des éléments pairs 's'.
Un élément T[i] est considéré pair s'il est divisible par 2

Calculer la somme des éléments impaires 's'.
Un élément T[i] est considéré impair s'il n'est pas divisible par 2

II.13. Exercice 12 (Décimal / Binaire / Octal)

1- Donner les codes des fonctions suivantes:

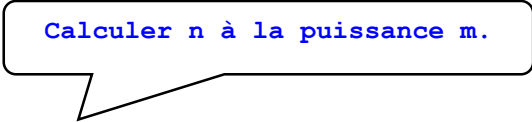
- a. **puissance(n,m :integer) :integer** ; qui permet de calculer et retourner le résultat de n^m (**n** a la puissance **m**).
- b. **decVersBin(n : integer) : longint**; qui permet de convertir le nombre décimal **n** donné en nombre binaire équivalent.
- c. **binVersDec(n : longint) : integer**; qui permet de convertir le nombre binaire **n** donné en nombre décimal équivalent.
- d. **decVersOct(n : integer) : longint**; qui permet de convertir le nombre décimal **n** donné en nombre octal équivalent.
- e. **octVersDec(n : longint) : integer**; qui permet de convertir le nombre octal **n** donné en nombre décimal équivalent.
- f. **Binaire (n : longint) :boolean** ; qui retourne « **true** » si **n** est un nombre binaire valide c'est-à-dire qu'il est représenté seulement par les chiffres 0 et 1.
- g. **octal (n : longint) :boolean** ; qui retourne « **true** » si **n** est un nombre octal valide c'est-à-dire qu'il est représenté par les chiffres de 0 à 7

2- En utilisant les fonctions précédentes, écrire un programme qui demande à l'utilisateur de saisir un nombre binaire et de le convertir en octal, puis il demande à l'utilisateur d'entrer un nombre octal et de le convertir en binaire.

Le programme doit prendre en compte tous les cas possibles y compris le cas d'erreur de saisi.

Solution

```
program Exercice2_12;
Uses crt;
var d:integer;
    b,o:longint;
function puissance(n,m:integer):integer;
var res,i:integer;
begin
    res:=1;
    for i:=1 to m do res:=res*n;
    puissance:=res;
end;
```



```
function decVersBin(n:integer):longint;
```

```
var
```

```
    bin,coef:longint;
```

```
    r:integer;
```

```
begin
```

```
    bin:=0;
```

```
    coef:=1;
```

```
    while (n<>0) do
```

```
        begin
```

```
            r:=n mod 2;
```

```
            n:=n div 2;
```

```
            bin:=bin+r*coef;
```

```
            coef:=coef*10;
```

```
        end;
```

```
    decVersBin:=bin;
```

```
end;
```

```
function binVersDec(n:longint):integer;
```

```
var
```

```
    r,coef,dec:integer;
```

```
begin
```

```
    dec:=0;
```

```
    coef:=0;
```

```
    while (n<>0) do
```

```
        begin
```

```
            r:=n mod 10;
```

```
            n:=n div 10;
```

```
            dec:=dec+r*puissance(2,coef);
```

```
            coef:=coef+1;
```

```
        end;
```

```
    binVersDec:=dec;
```

```
end;
```

Calculer le nombre 'bin' en base 2
équivalent à 'n' en base 10

Calculer le nombre 'dec' en base 10
équivalent à 'n' en base 2

```
function decVersOct (n:integer):longint;
```

```
var
```

```
    oct,coef:longint;
```

```
    r:integer;
```

```
begin
```

```
    oct:=0;
```

```
    coef:=1;
```

```
while (n<>0) do
```

```
    begin
```

```
        r:=n mod 8;
```

```
        n:=n div 8;
```

```
        oct:=oct+r*coef;
```

```
        coef:=coef*10;
```

```
    end;
```

```
    decVersOct:=oct;
```

```
end;
```

```
function octVersDec (n:longint):integer;
```

```
var
```

```
    r,coef,dec:integer;
```

```
begin
```

```
    dec:=0;
```

```
    coef:=0;
```

```
while (n<>0) do
```

```
    begin
```

```
        r:=n mod 10;
```

```
        n:=n div 10;
```

```
        dec:=dec+r*puissance (8,coef);
```

```
        coef:=coef+1;
```

```
    end;
```

```
    octVersDec:=dec;
```

```
end;
```

Calculer le nombre 'oct' base 8
équivalent à 'n' en base 10

Calculer le nombre 'dec' en base 10
équivalent à 'n' en base 8

```
function binaire(n:longint):boolean;
```

```
var
```

```
    res:boolean;
```

```
    r:integer;
```

```
begin
```

```
    res:=true;
```

```
    while(n<>0) and (res)do
```

```
        begin
```

```
            r:=n mod 10;
```

```
            n:=n div 10;
```

```
            if(r>=2)then res:=false;
```

```
        end;
```

```
    binaire:=res;
```

```
end;
```

```
function octal(n:longint):boolean;
```

```
var res:boolean; r:integer;
```

```
begin
```

```
    res:=true;
```

```
    while(n<>0) and (res)do
```

```
        begin
```

```
            r:=n mod 10;
```

```
            n:=n div 10;
```

```
            if(r>=8)then res:=false;
```

```
        end;
```

```
    octal:=res;
```

```
end;
```

```
begin
```

```
    {*forcer l'utilisateur à saisir un nombre binaire*}
```

```
    repeat
```

```
        write('Entrer un nombre Binaire SVP: ');
```

```
        readln(b);
```

```
    until binaire(b);
```

Un nombre 'n' est considéré binaire si tous les chiffres qui lui composent sont <2

Un nombre 'n' est considéré octal si tous les chiffres qui lui composent sont <8

```

o:=decVersOct(binVersDec(b));
writeln('Le nombre octal equivalent=',o);
{*forcer l'utilisateur à saisir un nombre octal*}
repeat
    write('Entrer un nombre octal SVP: '); readln(o);
until octal(o);
b:=decVersBin(octVersDec(o));
writeln('Le nombre binaire equivalent=',b);
Readkey ;
end.

```

II.14. Exercice 13 (Les anagrammes)

Donner le code de la fonction **dedans(str: String ; c :char) :boolean** ; qui retourne vraie si le caractère **c** existe dans la chaîne de caractères **str**.

Deux mots sont dites **anagrammatiques** s'ils contiennent les mêmes caractères en mêmes nombres. **Exemple** : « logarithme » est l'anagramme de « algorithme ».

Donner le code de la fonction **anagrammatiques (str1, str2: String) :boolean** ; elle retourne **True** si les deux chaînes **str1** et **str2** sont des anagrammes l'un de l'autre.

Ecrire le code qui demande à l'utilisateur de saisir 2 mots **ch1** et **ch2**, chacun à une longueur minimal de 3 caractères, et qui affiche le message 'Les deux mots sont anagrammatiques' si ils sont anagrammatiques.

Note : Les chaînes de caractères de type string en Pascal peuvent être assimilées à des tableaux de caractères. Comme pour ces derniers, il est possible de les indexer en précisant à la fin la position du caractère auquel on souhaite accéder. Si on considère la chaîne **str : String**, pour connaître le 3eme caractère, on écrira **str[3]** (le resultat est de Type **char**).

Solution

```

program Exercice2_13;
Uses crt;
var ch1,ch2: String;

    function dedans(str:string;c:char):boolean ;
    var res:boolean; i:integer;
    begin

```



```

res:=false;
i:=1;
while (i<=length(str) and (res=false)) do
    begin
        if (str[i]=c) then res:=true;
        i:=i+1;
    end;
dedans:=res;
end;

function anagrammatiques(str1,str2 :String) :boolean ;
var res:boolean; i:integer;
begin
    res:=true;
    if (length(str1)<>length(str2)) then res:=false
    else
        begin
            i:=1;
            while (i<=length(str2)) do
                begin
                    if (not dedans(str1, str2[i])) then
                        res:=false;

                    i:=i+1;
                end;
            end;
            anagrammes:=res;
        end;
end;

begin
    /*forcer l'utilisateur à saisir une chaine de 3 caractères au moins*/
    repeat
        write('Entrer la première chaine: ');
        readln(ch1);
        if (length(ch1)<3) then writeln('Erreur de saisie!');
    until (length(ch1)>=3);

```

Cette fonction retourne 'TRUE', S'il existe un caractère de la chaine 'str' = 'c'

Deux chaines 'str1' et 'str2' se considère anagrammatiques si tous les caractères de 'str2' existe dans la 'str1'

```
{*forcer l'utilisateur à saisir une chaine de 3 caractères au moins*}  
repeat  
    write('Entrer la deuxième chaine: '); readln(ch2);  
    if (length(ch2)<3) then writeln('Erreur de saisie!');  
until (length(ch1)>=3);  
if(anagrammatiques (ch1,ch2)) then  
    writeln('Les deux mots sont anagrammatiques')  
else writeln('Les deux mots ne sont pas des anagrammes');  
    readkey ;  
end.
```

Sachant que la fonction **length(str)** renvoie la longueur de la chaîne **str**.

II.15. Conclusion

En informatique lorsqu'un programme devient plus complexe il est nécessaire d'adopter une structuration modulaire, cette structuration en sous-programmes (Procédures et fonctions) permet de décomposer le programme en plusieurs modules. Chaque module peut également être décomposé en modules plus détaillés. A travers les exercices de ce chapitre, nous avons appris comment définir et utiliser nos propres procédures et fonctions et nous avons montré la nécessité de décomposer les problèmes à résoudre en plusieurs modules. Chaque sous-programme est alors traité séparément et l'intégration des différents modules conduit finalement à la solution globale du problème traité.

Chapitre

3

Les enregistrements et fichiers

III.1. Les enregistrements et fichiers (Rappel)

Un type structuré (ou enregistrement) consiste à définir de nouveaux types de données différents de ceux disponibles par défaut dans Pascal.

L'enregistrement est une structure de données qui représente un ensemble de propriétés hétérogènes. Chaque élément est appelé un champ.

Pour définir un nouveau type en pascal, nous utilisons le mot-clé « Type » suivi par un identificateur valide qui représente le nom de type que nous voulons créer et en spécifiant le nom et le type de chaque champ. La déclaration à la syntaxe suivante :

```
Type nouveauType=record
    champ1 : Type1;
    champ2 : Type2;
    ...
end ;
```

Les types structurés peuvent contenir d'autres types structurés ; un type peut avoir un niveau illimité de structuration.

III.2. Questions

Donner le code permettant de créer un nouveau type appelé **Employeur** qui se caractérise par les champs (Matricule, Nom, Prénom et Adresse), puis déclarer une variable **e** de type **Employeur**.

Donner l'ensemble d'instructions permettant de demander à l'utilisateur de saisir les informations de l'employeur **e**, puis donner l'instruction permettant d'afficher à l'écran les propriétés de l'employeur **e**.

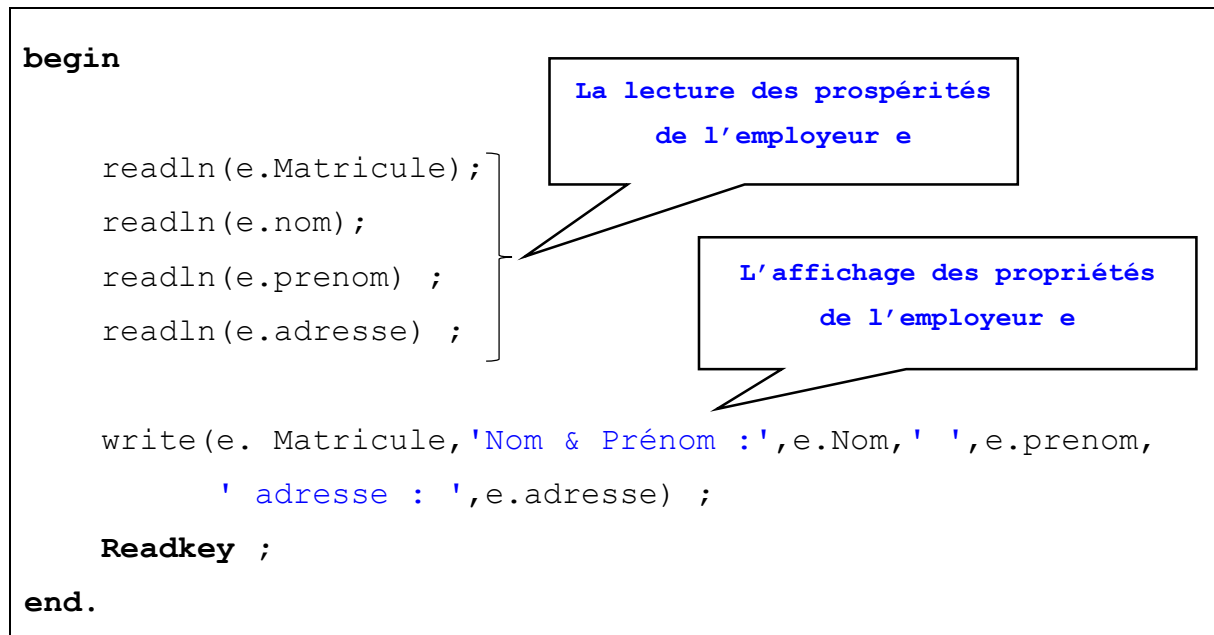
Réponses

```
program ExempleEnregistrement;
Uses crt ;
Type Employeur=record
    Matricule : integer;
    nom, prenom, adresse : String ;
end ;

var e: Employeur ;
```

La création d'un nouveau type nommée Employeur

Déclaration d'une variable e de type



Un fichier est une collection de données stockées sur un support de stockage. Pascal traite un fichier comme une séquence d'enregistrements, qui doivent être de type uniforme. Le type d'un fichier est déterminé par le type de l'enregistrement de base.

```

var fichier : file of type_de_base;

```

L'identifiant d'un fichier en Pascal se fait à travers :

Une représentation externe : c'est le nom du fichier pour le système de fichier du système d'exploitation considéré, par exemple : 'c:\etudiants.txt' .

Une représentation interne : c'est le nom du fichier connu par le programme, il est déclaré par le programme, par exemple : f .

L'association du fichier physique à sa représentation interne se fait a travers la procedure

```

Assign. Exemple : assign(f , 'c:\etudiants.txt');

```

Les principales méthodes qui manipulent sur les fichiers sont les suivantes :

La fonction eof(f) : boolean ; retourne vraie si nous pointons vers la fin du fichier.

La procédure assign(f , 'c:\etudiants.txt') ; permet d'attribue le nom 'f' au fichier 'c:\etudiants.txt'.

La procédure rewrite(f) ; permet d'effacer le contenu du fichier assigné à 'f' s'il existe, et de le créer sinon.

La procédure reset(f) ; permet d'ouvrir le fichier en mode ajout.

La procédure write(f,e) ; permet d'enregistrer l'étudiant e dans le fichier.

La procédure `read(f, e)` ; permet de lire depuis le fichier un étudiant `e`.

La procédure `close(f)` ; permet de fermer le fichier.

La procédure `seek(f, i)` ; permet de commencer la lecture à partir de la $i^{\text{ème}}$ enregistrement du fichier assigné à `'f'`.

La procédure `Truncate(f)` ; permet de couper à l'endroit actuel le contenu du fichier assigné à `'f'`.

La fonction `fileSize(f) : int` ; retourne le nombre d'enregistrement du fichier.

III.3. Exercice 1 (Gestion de scolarité)

Un étudiant se caractérise par les champs : Numéro d'inscription «NI», Nom, Prénom, cycle [licence, master ou doctorat] et niveau [1^{er} Année, 2^{ème} Année...]; Supposons que nous voulons créer un fichier « étudiantsST», constitué d'enregistrements représentant tous les étudiants inscrits actuellement à la faculté des sciences et technologie.

1. Proposer une structure de données permettant de gérer la liste des étudiants de la faculté des sciences et technologie.
2. Ecrire la procédure **creerFichier()** ; qui permet de créer physiquement le fichier « étudiantsST » sur un disque local.
3. Ecrire une procédure permettant d'enregistrer **n** étudiants dans le fichier « étudiantsST ».
4. Donner le code de la procédure **etudiantsDeNom(C : String)** ; permettant d'afficher la liste des étudiants ayant le nom de famille **C**.
5. Ecrire une fonction **compter (n :integer ; C :String) :integer** ; permettant de compter le nombre d'étudiants inscrits en $n^{\text{ème}}$ Année du cycle **C**.

Solution

```
program exercice3_1;
Uses crt ;
Type Etudiant=record
    NI : integer;
    nom, prenom, cycle: String[20] ;
    niveau :integer ;
end ;
(*la représentation externe du fichier *)
```

Les paramètres de
l'enregistrement
'Etudiant'

```
Const fichier='f:\etudiants.txt';
var ch,nb,v:integer; c:string;
  procedure creerFichier();
  var f :file of Etudiant ;{la représentation interne du fichier}
  begin
  {*assign permet d'attribuer le nom 'f' au fichier 'c:\etudiants.txt'*}
    assign(f ,fichier);
    reset(f); {* permet d'ouvrir le fichier en mode ajout *}
    close(f); {* permet de fermer le fichier *}
    writeln('Le fichier a été crée:') ;
  end;
  procedure enregistrerEtudiants(n:integer);
  var f :file of Etudiant ;
      i :integer ; e:Etudiant ;
  begin
    assign(f ,fichier);
    reset(f);
    for i :=1 to n do
      begin
        writeln('Entrer N°:',i);
        write('Numero d'inscription:');
        readln(e.NI);
        write('Nom:');
        readln(e.nom);
        write('Prenom:');
        readln(e.prenom) ;
        write('e.cycle:') ;
        readln(e.cycle) ;
        write('e.Niveau: ') ;
        readln(e.niveau) ;
        Write(f ,e);{* enregistrer l'étudiant e dans le f *}
      end ;
    close(f);
  end;
```

La saisie des informations des 'n' étudiants

```

procedure etudiantsDeNom(C : String) ;
var f :file of Etudiant ;
    e:Etudiant ;
begin
    assign(f ,fichier);
    reset(f);
    nb:=0; {* nombre des étudiants trouvés initialisé à 0 *}
    while not eof(f) do
        begin
            read(f ,e);
            if(e.Nom=c) then
                begin
                    writeln(e.NI,' Nom & Prenom :',e.Nom,
                        ' ',e.prenom,' Cycle : ',e.cycle,
                        ' Niveau : ',e.niveau) ;
                    nb:=nb+1;
                end;
            end;
        end;
    writeln('Nombre d'etudiants trouvees est ', nb);
    close(f);
end;

function compter (n :integer ; C :String) :integer ;
var f :file of Etudiant ;
    e:Etudiant ; nbr :integer ;
begin
    assign(f ,fichier);
    reset(f);
    nbr:=0; {* nombre des étudiants trouvés initialisé à 0 *}
    while not eof(f) do
        begin
            read(f ,e);
            if(e.cycle=c) and (e.niveau=n) then nbr:=nbr+1;
        end;
    end;

```

Pour chaque étudiant 'e' du fichier 'f'
Incrémenter 'nb' et afficher les informations
de l'étudiant 'e' si son nom =c.

Pour chaque étudiant 'e' du fichier 'f'
Incrémenter 'nbr' si l'étudiant 'e'
appartient au cycle 'c' et si son
niveau = n.


```

        close(f );
        compter:=nbr;
    end;
procedure supprimerEtudiant(NI:integer);
var f :file of Etudiant ;
    e:Etudiant;
    tr:boolean; {* tr =TRUE si l'étudiant trouvé *}
    i:integer; {*l'indice de l'enregistrement à supprimée*}
begin
    assign(f ,fichier);
    reset(f);
    tr:=false; {* l'étudiant n'est pas trouvé *}
    i:=1;
    while not eof(f) and not tr do
        begin
            read(f ,e);
            if (e.NI=NI) then tr:=true;
            else i:=i+1;
        end;
        if tr then {* si l'etudiant trouvé*}
            begin
{*commencer la lecture à partir du dernier enregistrement du fichier*}
                Seek(f,FileSize(f)-1);
                Read(f,e); {*e c'est le dernier etudiant dans f*}
{*commencer la lecture à partir de la ième enregistrement du fichier*}
                Seek(f,i);
                Write(f,e); {*enregistrer e à l'indice 'i'*}
{*commencer la lecture à partir de dernier enregistrement du fichier*}
                Seek(f, FileSize(f)-1);
{*couper à l'endroit actuel le contenu du fichier*}
                Truncate(f);
                writeln('etudiant Supprimé');
            end
        else write('Numero d'inscription introuvable..');
    
```

Pour chaque étudiant 'e' du fichier 'f'
Etudiant trouvé si le numéro
d'inscription de 'e'=NI

```
        Close(f);
    end;

begin
    writeln('Selectionner :');
    writeln('  1: Enregistrer les etudiants');
    writeln('  2: Supprimer un etudiant');
    writeln('  3: Afficher les etudiants Par Cycle');
    writeln('  4: Compter le nombre d''etudiants');
    writeln('  5: regenerer le fichier');
    write('Entrer votre choix:');
    readln(ch);

    case ch of
        1:
            begin
                write('nombre d''etudiants à enregistrer:');
                readln(nb);
                enregistrerEtudiants(nb);
                write('Enregistrement Terminée');
            end;
        2:
            begin
                writeln('Supression d''un etudiant..');
                write('numero d''inscription:');
                readln(nb);
                supprimerEtudiant(nb);
            end;
        3:
            begin
                write('Entrer le Nom de l''etudiant:');
                readln(c);
                etudiantsDeNom(c);
            end;
    end;
end;
```

Un menu qui s'affiche à l'utilisateur

```
    4: begin
        write('Cycle:');
        readln(c);
        write('Niveau:');
        readln(nb);
        nb:=compter(v,c);
        writeln('Le nombre d''etudiants est = ',nb);
    end;
    5: creerFichier();
    else writeln('execution terminee.. ');
end;
readkey ;
end.
```

III.4. Exercice 2 : (La gestion des ventes automobiles)

Une entreprise commerciale souhaite améliorer la gestion des ventes automobiles. Chaque véhicule se caractérise par son Numéro de châssis, son Modèle, sa Couleur et son prix.

1. Proposer une structure de données permettant de gérer le processus des ventes automobiles.
2. Ecrire la procédure **creerFichier()** ; qui permet de créer le fichier **AUTOMOBILES.TXT** s'il n'existe pas.
3. Ecrire la fonction **existVehicule(NC :String) : boolean** ; qui retourne vraie si le véhicule ayant le numéro de châssis **NC** existe dans le fichier **AUTOMOBILES.TXT**.
4. Ecrire la procédure **ajouterVehicule()** ; permettant d'insérer un nouveau véhicule à la fin du fichier **AUTOMOBILES.TXT**.

NB : Il est impossible d'ajouter un véhicule avec un numéro de châssis déjà attribué à une autre voiture dans le fichier **AUTOMOBILES.TXT**.

5. Ecrire la procédure **ajouterVehicules(n :integer)** ; qui permet d'enregistrer « n » véhicules dans le fichier **AUTOMOBILES.TXT** à travers la procédure précédente.
6. Ecrire la procédure **afficherVehicules()** ; permettant d'afficher tous les véhicules avec ses ordres d'enregistrements dans le fichier.
7. Ecrire la procédure **afficherVehicules()** ; qui permet d'afficher les véhicules enregistrés dans le fichier **AUTOMOBILES.TXT**.

8. Ecrire la procédure **supprimerVehicule(indice:Integer)** ; permettant de supprimer le véhicule enregistré à la position « **indice** » dans le fichier AUTOMOBILES.TXT tout en conservant l'ordre d'enregistrement des véhicules.
9. Ecrire la procédure **modifierVehicule(indice :String)** ; permettant de modifier les propriétés d'un véhicule enregistrée dans le fichier AUTOMOBILES.TXT à la position « **indice** ».
10. Ecrire la procédure **chercherVehicule(M:String)** ; qui permet d'afficher la liste des véhicules ayant le model **M** avec ses ordres d'enregistrements dans le fichier.
11. Ecrire un programme permettant d'exécuter au choix les fonctionnalités (1: Ajouter véhicules, 2: Afficher les véhicules, 3: Modifier véhicule, 4: supprimer véhicule et 5: Chercher véhicule) en utilisant les procédures et les fonctions précédentes.

Solution

```

program Exercice3_01;
Uses crt ;
Type Auto=record
    NC, Model,Couleur : String[17];
    prix : real ;
end ;
Const fichier='f:\AUTOMOBILES.TXT';
var nb, ch, ind :integer ;
    model:String;
    f :file of Auto ;
procedure creerFichier() ;
begin
    assign(f,fichier);
    {I-}
    reset(f);
    {I+}
    if (IORESULT <> 0) then {S'il existe une erreur *}
        begin
            writeln('Le fichier est introuvable..');
{effacer le contenu du fichier 'f' s'il existe, et de le créer sinon*}
            rewrite(f);

```

Les caractéristiques de l'enregistrement 'Auto'

En cas d'erreur de lecture pendant l'exécution de la procédure reset(f), les directives **{I-}** et **{I+}** modifient la valeur du variable système IORESULT

```

        writeln('Le fichier a été créée..');
    end
    else { * s'il n'existe pas d'erreurs *}
        writeln('Le fichier ',fichier,' Trouvé');
    close(f);
end;
function existVehicule(NC :String) : boolean ;
var tr:boolean; v:Auto;
begin
    tr:=false; { * vehicule intialisé non trouvé*}
    assign(f,fichier);
    reset(f);
    while not eof(f) and not tr do
        begin
            read(f,v);
            if (v.NC=NC) then
                tr:=true;
            end;
        close(f);
        existVehicule:= tr;
    end;
procedure ajouterVehicule();
var v:Auto;
begin
    write('Entrer le N de chassis:');
    repeat
        readln(v.NC);
        if (existVehicule(v.NC)) then
            write('Ce numéro existe; entrer un autre:');
        until not existVehicule(v.NC);
    write('Entrer le Model:');
    readln(v.Model);
    write('Entrer la couleur');

```

Véhicule est considéré trouvé (tr=True) si le numéro de chassis du véhicule v =NC

Pour ajouter un nouveau véhicule, il faut que son numéro de chassis ne se figure pas dans le fichier des véhicules

Entrer les autres caractéristiques du véhicule 'v' à ajouter

```

    readln(v.Couleur);
    write('Entrer le Prix:');
    readln(v.prix);
    assign(f,fichier);
    reset(f);
Seek(f, FileSize(f)); {*pointer vers la fin du fichier 'f' *}
    write(f,v);{*ajouter le véhicule 'v' à cet endroit dans f*}
    close(f);
end;

procedure ajouterVehicules(n :integer) ;
var v:Auto;
    i:integer;
begin
    for i:=1 to n do
        saisirVehiculeInfo();
    end;

procedure afficherVehicules() ;
var v:Auto;
    indice:integer;
begin
    assign(f,fichier);
    reset(f);
    indice:=0;
    while not eof(f) do
        begin
            read(f,v);
            writeln(indice,': Model: ',v.Model:10,
                ' Couleur: ',v.couleur:11,
                ' PRIX:',v.prix:4:2,
                ' Chassi N:',v.NC);
            indice:=indice+1;
        end;
end;

```

Ajouter 'n' véhicules

Pour chaque vehicule 'v' du fichier 'f'
Afficher les caractéristiques de ce véhicule

```

        writeln('le nombre de vehicules est: ',FileSize(f));
        close(f);
    end;
    procedure supprimerVehicule(i:integer);
    var a:Auto;
    begin
        Assign(f, fichier);
        Reset(f);
        if (i < FileSize(f)) then
            begin
                while(indice+1<FileSize(f)) do
                    begin
                        Seek(f, i+1);
                        Read(f, a);
                        Seek(f, i);
                        Write(f, a);
                        i:=i+1;
                    end ;
                    {*pointer vers le dernier élément du fichier 'f' *}
                    Seek(f, FileSize(f)-1);
                    {*couper à l'endroit actuel le contenu du fichier*}
                    Truncate(f);
                end
            else writeln('l'indice ',i,' n'existe pas');
            Close(f);
        end;
    procedure modifierVehicule(i:integer);
    var v:Auto;
    begin
        Assign(f, fichier);
        Reset(f);
        if (i<FileSize(f)) then
            begin
                writeln('Entrer le N de chassis:');

```

FileSize(f) renvoie le nombre d'enregistrement du fichier 'f'

Si l'indice i < nombre d'enregistrement
Décaler tous les enregistrements qui le suivent vers l'index inférieur

Si l'indice i < nombre d'enregistrement
Demander à l'utilisateur de saisir les caractéristiques du véhicule à modifier

Il faut que son numéro de châssis ne se figure pas dans le fichier des véhicules

```

repeat
  readln(v.NC);
  if(existVehicule(v.NC)) then
    write('Chassis exist; entrer un autre:');
  until not existVehicule(v.NC);
  write('Entrer le Model:');
  readln(v.Model);
  write('Entrer la couleur');
  readln(v.Couleur);
  write('Entrer le Prix:');
  readln(v.prix);
  Seek(f, i); {*pointer vers le ième indice du fichier *}
  Write(f, v); {*ajouter le véhicule v à cet endroit*}
end
else writeln('l''indice ',indice,' n''existe pas');
Close(f);
end;
procedure chercherVehicule(M:String);
var indice:integer; v:Auto;
begin
  assign(f,fichier);
  reset(f);
  indice:=0;
  while not eof(f) do
    begin
      read(f,v);
      if(v.model=M) then
        writeln(indice,': Model: ',v.Model:10,
          ' Couleur: ',v.couleur:11,
          ' PRIX:',v.prix:4:2,
          ' Chassi N:',v.NC);
        indice:=indice+1;
      end;
    end;
  end;

```

Pour chaque vehicule 'v' du fichier 'f'
Afficher les caractéristiques du
véhicules ayant le modèle 'M'


```
writeln('le nombre de vehicules est: ',FileSize(f));
close(f);
end;
begin
  creerFichier();
  repeat
    writeln;
    writeln('Selectionner :');
    writeln('  1: Ajouter vehicules');
    writeln('  2: Afficher les vehicules');
    writeln('  3: Modifier vehicule');
    writeln('  4: supprimer vehicule');
    writeln('  5: Chercher vehicule');
    writeln('  6: regenerer le fichier');
    write('Entrer votre choix:'); readln(ch);
    case ch of { * selon le choix de l'utilisateur *}
      1:begin
          write('entrer Le nombre de vehicules :');
          readln(nb);
          ajouterVehicules(nb);
        end;
      2:afficherVehicules();
      3:begin
          write('Entrer l''indice du vehicule:');
          readln(ind);
          modifierVehicule(ind);
        end;
      4:
        begin
          write('Entrer l''indice du vehicule:');
          readln(ind);
          supprimerVehicule(ind);
        end;
    end;
  end;
end;
```

Un menu qui s'affiche à l'utilisateur

```
5:
    begin
        write('Entrer le model a chercher: ');
        chercherVehicule(model);
    end;
6:
    Begin
        assign(f,fichier);
        rewrite(f);
        close(f);
        writeln('le fichier a ete regeneree');
    end ;
    else writeln('execution terminee.. ');
end;
until (ch>6);
readkey ;
end.
```

Par défaut en cas d'erreur système le programme s'arrête, mais avec l'utilisation des directives **{!-}** et **{!+}** le programme ne s'arrête pas et modifie la valeur du variable system **IOResult**.

III.5. Exercice 3 (La gestion des Clients)

La même entreprise veut organiser les transactions avec ses clients dans le fichier **TRANSACTIONS.TXT**, chaque transaction se caractérise par le Nom, le prénom du client, son Numéro de téléphone et le Numéro de châssis du véhicule vendu.

Note : Un véhicule se considère vendu si son numéro de châssis est associé à un client dans le fichier des transactions.

1. Proposer une structure de données qui permet de gérer les transactions de cette société.
2. Ecrire la procédure **creerFichierTransaction()** ; qui permet de créer le fichier **TRANSACTIONS.TXT** s'il n'existe pas.
3. Ecrire une fonction **vendue (NC : String) : boolean** ; qui retourne vraie si le numéro de châssis NC apparait dans le fichier des transactions.

4. Ecrire la procédure **chercherVehicule(C, M : String)** ; qui affiche la liste des véhicule invendu avec la couleur **C** et le model **M**.
5. Ecrire la fonction **chercheNumChassis(indice :Integer) :String** ; qui renvoie le numéro de châssis du véhicule enregistré à la position « **indice** » dans le fichier **AUTOMOBILES.TXT**. Cette fonction retourne RIEN si cette véhicule soit vendue ou l'indice n'existe pas.
6. Ecrire la procédure **ajouterTransaction(NC :String)** ; qui demande à l'utilisateur de saisir les informations du client et d'associé le Numéro de châssis **NC** à ce client dans le fichier des transactions.
7. Ecrire la procédure **annulerTransaction(indice :Integer)** ; qui permet de supprimer la transaction de vente avec le client enregistré à la position « **indice** » dans le fichier des transactions.
8. Ecrivez la procédure **afficherTransactions ()** ; qui affiche les transactions de vente du dernier enregistrement au premier.
9. Ecrire un programme permettant d'exécuter au choix les fonctionnalités (1 : Ajouter une transaction de vente 2 : Annuler une transaction de vente 3 : Chercher véhicules selon une couleur et un model données, et 4 : Afficher toutes les transactions) en utilisant les procédures et les fonctions précédentes.

Solution

```

program Exercice3_03;
Uses crt ;
Type Auto=record
    NC, Model, Couleur : String[17];
    prix : real ;
end ;
Transaction=record
    Nom, Prenom, NC :
    Tel : Integer ;
end ;
Const {*la représentation externe des 2 fichiers *}
    fichier='f:\TRANSACTIONS.TXT';
    fichierV='f:\AUTOMOBILES.TXT';

```

Les caractéristiques de l'enregistrement 'Auto'

Les caractéristiques de l'enregistrement 'Transaction'

Les caractéristiques de l'enregistrement 'Transaction'

```

var ch,ind :integer ;
    chassis, model, couleur: String;
    f :file of Transaction;
    fv :file of Auto ;

procedure creerFichierTransaction() ;
begin
    assign(f,fichier);
    {$I-}
        reset(f);
    {$I+}
    if IORESULT <> 0 then { *S'il existe une erreur*}
        rewrite(f);
    close(f);
end;

function vendue(NC :String) : boolean ;
var tr:boolean;
    t:Transaction;
begin
    tr:=false; { * véhicule intialisé non vendu *}
    assign(f,fichier);
    reset(f);
    while not eof(f) and not tr do
        begin
            read(f,t);
            if(t.NC=NC) then tr:=true;
        end;
    close(f);
    vendue:= tr;
end;

procedure chercherVehicule(C,M:String);
var indice:integer;
    v:Auto;
begin

```

La représentation interne des 2 fichiers

Créer le fichier, s'il existe une erreur pendant l'exécution de la procédure reset(f)

Un véhicule ayant le numéro de chassis NC se considère vendu, Si ce NC apparait dans le fichier des transactions 'f'

```

assign(fv,fichierV);
reset(fv);
indice:=0;
while not eof(fv) do
  begin
    read(fv,v);
    if(v.model=M) and (v.couleur=C)
      and(not vendue(v.NC)) then
      writeln(indice,': Model: ',v.Model:10,
        ' Couleur: ',v.couleur:11,' PRIX:',v.prix:4:2,
        ' Chassi N:',v.NC);
      indice:=indice+1;
    end;
  close(fv);
end;

function chercheNumChassis(indice:integer):String;
var v:Auto; NC:String;
begin
  assign(fv,fichierV);
  reset(fv);
  if(indice<FileSize(fv)) then
    begin
      seek(fv, indice);
      read(fv,v);
      if(vendue(v.NC)) then
        begin
          writeln('ce vehicule est vendu..');
          NC:='RIEN';
        end
      else
        NC:=v.NC;
      end
    end
  else

```

Pour chaque vehicule 'v' du fichier fv
Afficher les caracteristiques des
vehicules invendus et ayant le modele
'M' et la couleur 'C'

Pointer vers le vehicule ayant
l'indice i si i<nombre de
vehicules enregistres

NC := 'RIEN' si le vehicule est vendu
Et prends le numero de chassis du
vehicule ayant l'indice 'i' sinon

```

        begin
            writeln('erreur: indice introuvable');
            NC:='RIEN';
        end;
    close(fv);
    rechercheNumChassis:=NC;
end;

procedure ajouterTransaction(NC:String) ;
var
    t:Transaction;
begin
    t.NC:=NC;
    writeln('Entrer le Nom du client:');
    readln(t.Nom);
    writeln('Entrer le Prenom du client:');
    readln(t.Prenom);
    writeln('Entrer le Numero de Tel du client:');
    readln(t.tel);
    assign(f,fichier);
    reset(f);
    Seek(f, FileSize(f)); {*pointer vers la fin du fichier f*}
    write(f,t); {*ajouter la transaction 't' à cet endroit*}
    close(f);
end;

procedure annulerTransaction(i:integer);
var
    t:Transaction;
begin
    Assign(f, fichier);
    Reset(f);
    if (i < FileSize(f)) then
        begin
            while (i+1<FileSize(f)) do

```

NC := 'RIEN' si l'indice 'i' > nombre de véhicules enregistrés

Saisir les informations de la transaction 't'

Si l'indice i < nombre d'enregistrement
Décaler tous les enregistrements qui le suivent vers l'index inférieur

```
begin
    Seek(f, i+1);
    Read(f, t);
    Seek(f, i);
    Write(f, t);
    i:=i+1;
end;
{*pointer vers le dernier élément du fichier 'f' *}
Seek(f, FileSize(f)-1);
{*couper à l'endroit actuel le contenu du fichier*}
Truncate(f);
writeln('Transaction supprimée');
end
else writeln('l'indice ',i,' n'existe pas');
Close(f);
end;

procedure afficherTransactions() ;
var t:transaction; i:integer;
begin
    assign(f, fichier);
    reset(f);
    i:=FileSize(f)-1;
    while (i>=0) do
        begin
            seek(f, i);
            read(f, t);
            writeln(indice, ': Chassis: ', t.NC,
                ' Nom: ', t.Nom, ' Prenom: ', t.prenom,
                ' tel: ', t.tel);
            i:=i-1;
        end;
        writeln('le nombre de transactions: ', FileSize(f));
        close(f);
    end;
end;
```

Afficher les transactions de la dernière vers la première

```
begin
  creerFichierTransaction();
  repeat
    writeln('Selectionner :');
    writeln('  1: Ajouter transaction');
    writeln('  2: Annuler une transaction');
    writeln('  3: chercher vehicule invendu');
    writeln('  4: liste de transactions');
    writeln('  5: regenerer le fichier');
    write('Entrer votre choix:');
    readln(ch);
    case ch of { * selon le choix de l'utilisateur *}
      1:
        begin
          write('Entrer l''indice du vehicule:');
          readln(ind);
          chassis:=chercheNumChassis(ind);
          if(chassis<>'RIEN') then
            begin
              writeln('Chassus N: ',chassis);
              ajouterTransaction(chassis);
              writeln('Transaction ajoutée');
            end;
          end;
        2:
          begin
            write(l''indice de la transaction:');
            readln(ind);
            annulerTransaction(ind);
          end;
        3:
          begin
            write('Entrer le model: ');
```

Un menu qui s'affiche à l'utilisateur


```
        readln(model);
        write('Entrer la couleur: ');
        readln(couleur);
        chercherVehicule(couleur,model);
    end;
4:
    begin
        writeln('Liste de transactions: ');
        afficherTransactions();
    end;
    else writeln(erreur de saisie.. ');
end;
until (ch>5);
readkey ;
end.
```

III.6. Exercice 4 (Les statistiques)

La même entreprise veut enregistrer les statistiques de ventes de ses véhicules dans le fichier **STATISTIQUES.TXT**. Chaque enregistrement de ce fichier se caractérise par le Model, le Nombre des véhicule total, le Nombre de véhicules vendus et les revenus collectés de ce modèle de véhicule.

1. Proposer une structure de données qui permet de gérer les statistiques de ventes.
2. Ecrire la fonction **nbrVehicule(M:String):Integer** ; permettant de compter le nombre de véhicules de model **M**.
3. Ecrire la fonction **venduesType(M:String):Integer** ; permettant de compter le nombre de véhicules de model **M** vendus.
4. Ecrire la procédure **genererStatistiques()** ; qui permet d'enregistrer les statistiques de ventes de tous les type de véhicules dans le fichier **STATISTIQUES.TXT**.
5. Ecrire un programme qui permet d'afficher les statistiques de vente pour tous les types de véhicules.

Solution

```

program Exercice3_03;
Uses crt ;
Type Stat=record
    Model:String[17];
    NVT, NVV:integer;
    Revenus:real;
end;
Auto=record
    NC, Model,Couleur : String[17]; prix : real ;
end ;
Transaction=record
    Nom, Prenom, NC : String[17];
    Tel : Integer ;
end ;
Const fichierT='f:\TRANSACTIONS.TXT';
    fichierV='f:\AUTOMOBILES.TXT';
    fichierS='f:\STATISTIQUES.TXT';
var fs :file of Stat;{*Représentation interne du fichier Stat*}
function vendue(NC :String) : boolean ;
var tr:boolean; t:Transaction; ft :file of Transaction ;
begin
    tr:=false;
    assign(ft,fichierT);
    reset(ft);
    while not eof(ft) and not tr do
        begin
            read(ft,t);
            if(t.NC=NC) then tr:=true;
        end;
    close(ft);
    vendue:= tr;
end;

```

Les caractéristiques de l'enregistrement 'Stat'

Les caractéristiques de l'enregistrement 'Auto'

Les caractéristiques de l'enregistrement 'Transaction'

La représentation externe des 3 fichiers

Un véhicule ayant le numéro de chassis NC se considère vendu, Si ce NC apparait dans le fichier des transactions 'ft'

```
function statistiqueModel(M:String):Stat;
```

```
var S:Stat;
```

```
  fv :file of Auto ;
```

```
  v:Auto;
```

```
begin
```

```
  S.Model:=M;
```

```
  S.NVT:=0;
```

```
  S.NVV:=0;
```

```
  assign(fv,fichierV);
```

```
  reset(fv);
```

```
  while not eof(fv) do ;
```

```
    begin
```

Pour chaque véhicule du fichier 'fv'

```
      read(fv,v);
```

```
      if (v.Model=M) then
```

Compter le nombre de véhicules totales ayant le modèle 'M'

```
        begin
```

```
          S.NVT:=S.NVT+1;
```

```
          if (vendue(v.NC)) then
```

```
            begin
```

Compter le nombre de véhicules vendus ayant le modèle 'M'.
Et les revenus de ce modèle

```
              S.NVV:=S.NVV+1;
```

```
              S.Revenus:=S.Revenus+v.prix;
```

```
            end;
```

```
          end;
```

```
    end;
```

```
  close(fv);
```

```
  statistiqueModel:=S;
```

```
end;
```

```
function comptee(M :String) : boolean ;
```

```
{*tr=True signifié que les véhicules ayant le modèle M sont comptabilisé}
```

```
var tr:boolean ; S:Stat;
```

```
begin
```

```
  tr:=false; {*Représentation interne du fichier Stat*}
```

```
  assign(fs,fichierS);
```

```
  reset(fs);
```

```

while not eof(fs) and not tr do
    begin
        read(fs,S);
        if(S.Model=M) then tr:=true;
    end;

close(fs);
comptee:= tr;
end;

procedure ajouterStatistique(S:Stat) ;
begin
    assign(fs,fichierS);
    reset(fs);
    Seek(fs, FileSize(fs));{*pointer vers la fin du fichier fs}
    {*ajouter les statistiques 'S' à cet endroit dans fs*}
    write(fs,S);
    close(fs);
end;

procedure extraireStatistiques();
var
    fv :file of Auto ;
    v:Auto;
    S:Stat;
    indice:integer;
    model:String;
begin
    assign(fv,fichierV);
    reset(fv);
    indice:=0;
    writeln(FileSize(fv), ' vehicules');
    while (indice<FileSize(fv)) do
        begin
            seek(fv,indice);
            read(fv,v);

```

Si le modele 'M' ne se figure pas dans le fichier des statistiques tr rest FALSE sinon elle devient TRUE

Afficher le nombre de véhicules enregistrés dans le fichier 'fv'

Comptabiliser les véhicules ayant le modèle 'm' s'ils ne sont pas comptabilisés.

```

        if(not Comptee(v.model)) then
            begin
                m:=v.Model;
                close(fv);
                S:=statistiqueModel(m);
                ajouterStatistique(S);
                reset(fv);
            end;
            indice:=indice+1;
        end;
    close(fv);
end;

procedure afficherStatistiques() ;
var
    indice:integer;
    S:Stat;
begin
    assign(fs,fichierS);
    reset(fs);
    indice:=1;
    while not eof(fs) do
        begin
            read(fs,S);
            writeln(indice:2,': Model: ',S.Model:17,
                ' Total/vendus:',S.NVT,'/',S.NVV,
                ' Revenus:',S.Revenus:0:2);
            indice:=indice+1;
        end;
    close(fs);
end;

begin
    assign(fs,fichierS);
    rewrite(fs);

```

Afficher les statistiques de chaque modèle comptabilisé dans le fichiers des statistiques 'fs'

```
close(fs);  
extraireStatistiques();  
afficherStatistiques();  
Readkey;  
end.
```

III.7. Conclusion

Un fichier est un ensemble de données structuré en unités d'accès appelées enregistrements ou articles qui sont tous de même type ; les fichiers nous permettront de manipuler les enregistrements et les stocker sur des supports physiques (disques durs, CDs...Etc). Un enregistrement est un type de données défini par le programmeur, il permet de grouper un nombre fini d'éléments de types éventuellement différents sous un nom commun. Les éléments qui composent un enregistrement sont appelés champs. Un champ d'enregistrement s'utilise comme une variable du même type (Char, String, Boolean, Integer,...ect). Les exercices présentés dans ce dernier chapitre permettent aux étudiants de manipuler les données et de les stocker sur des supports externes en se basant sur la programmation modulaire.

Références bibliographiques

- [1] Zakaria, Smahi. "Algorithmique et programmation en Pascal et Fortran: Cours et exercices corrigés. University of Sciences and Technology of Oran, 2021.
- [2] Bouziane, Hafida. "Algorithmique et Structures de Données II (ASD2). University of Sciences and Technology of Oran, 2021.
- [3] Cormen, Thomas H., et al. Introduction to algorithms. MIT press, 2009.
- [4] Aho Alfred, V., et al. Data structures and algorithms. USA: Addison-Wesley, 1983.
- [5] Chabert, Jean-Luc. Histoire d'algorithmes: du caillou à la puce. Belin, 1994.
- [6] Charles, Henri-Pierre. Initiation à l'informatique: programmation, algorithmique, architectures. Eyrolles, 1999.