TD-7
fonctions et procedures

Exercice 1 :

Ecrire un programme qui lit une matrice carrée d'entiers d'ordre 4 et
- Pour chaque $i \leq 4$ affiche la somme des élements de la ligne i et de la colonne i.
- Affiche la somme de tous les élements de la matrice.

Pour cela, définir et utiliser les fonctions:

- **init**, qui prends comme argument une matrice A, et la rempli avec 16 entiers entrés par l'utilisateur.
- **somme-l**, qui prends comme argument une matrice et une indice de ligne et donne la somme des élements de la ligne.
- **somme-c**, qui prends comme argument une matrice et une indice de colonne et donne la somme des élements de la colonne.

Exercice 2 :

Ecrire une fonction $max(x, y)$ qui retourne le plus grand elements de x ou de y lorsque ceux ci sont des entiers.
Ecrire une fonction $max(t, i, j)$, où t est un tableau d'entiers de dimension dim, et qui donne le maximum du tableau entre les positions i et j.
Ecrire une fonction $position(t, x)$ qui retourne la plus petite position de t qui contienne x. Elle retourne -1 si x n'est pas dans le tableau.
Ecrire une procédure $echange(t, i, j)$ qui echange les elements de position i et j dans t.
Ecrire un programme qui trie un tableau, en selectionnant l'element maximum d'un tableau, qui le place en derniere position, puis selectionne l'element maximum du tableau restant a trier et le place en avant derniere position etc...

Exercice 3 : Un jeu de Pile ou Face.

On considère un plateau de jeu de 7 cases, chacune contenant une pièce. Initialement elle sont toutes à Pile. Le joueur doit, s'il veut gagner atteindre une position ou toutes les cases sont sur Face. Pour cela il doit choisir deux pièces quelconques et les retourner.
On modélise le plateau de jeu par un tableau de dimention 7, remplis avec des 0 et des 1, 0 pour pile et 1 pour face. On désire faire un programme pour jouer à ce jeu.

- Ecrire une procédure $init(plateau)$ qui permet d'initialiser le plateau de jeu.
- Ecrire une procédure $affiche(plateau)$ qui traduise les 0 en Pile et les 1 pour Face.
- Ecrire une fonction $gagne(plateau)$ qui retourne 0 ou 1 selon que le joueur a gagné la partie ou non.
- Ecrire une fonction $coupValide(i, j)$ qui retourne 0 ou 1 si le coup “retourner les pièces en positions i et j” est un coup autorisé.
- Ecrire une procédure $flip(plateau, i, j)$ qui retourne les pièces i et j. Si le coup n'est pas autorisé le plateau reste inchangé.
- Ecrire un programme pour le jeu complet, qui demande au joueur de continuer tant qu'il n'a pas gagné.

Notes et corrigé pour le TD-7

Corrigé exercice 1 :

```
#include<iostream.h>

const int Dim=4;

typedef int vecteur[Dim];
typedef vecteur matrice[Dim];

void init(matrice& a) {
int i,j;
for(i=0;i<Dim;i++){
  for(j=0;j<Dim;j++){
cout<<"\nmat["<<i<<","<<j<<"]=";
cin>>a[i][j];
  }
}
}

int somme-l (matrice a,int i){
int j,res=0;
for (j=0;j<Dim;j++){
  res+=a[i][j];}
return res;
}

int somme-c (matrice  a,int i){
int j,res=0;
for (j=0;j<Dim;j++){
  res+=a[j][i];}
return res;
}

void main(){

int i,somme=0;
matrice b;

init(b);
for(i=0;i<Dim;i++){
cout<<"\nsomme ligne "<<i<<"="<<somme-l(b,i);
}

for(i=0;i<Dim;i++){
cout<<"\nsomme colonne "<<i<<"="<<somme-c(b,i);
}

for(i=0;i<Dim;i++){
somme+= somme-l(b,i);
}

cout<<"\nsomme matrice  ="<<somme<<"\n";
}
```

Corrigé exercice 2 :

```
constante  dim = 10
type vecteur = tableau de dim entier
```

```
fonction max(x,y : entier) : entier
variable rep : entier
debut max
  si (x>y) alors rep <- x
           sinon rep <- y
  fin si
  retourner rep
fin max

// on suppose i<j
fonction max(reference t:vecteur,i,j : entier) : entier
variable k,rep : entier
debut max
   rep <- t[i]
   pour k de i+1 a j faire
     rep <- max(rep,t[k])
   finpour
   retourner rep
fin max

fonction position(reference t : vecteur, x : entier)
variable i, rep : entier
debut position
i <- 1
tant que (i<= dim et t[i] <> x) faire
     i <- i+1
fin tantque

si (i=dim+1) alors rep <- -1
             sinon rep <- i
fin si

retourner rep
fin position


procedure echange(reference t:vecteur, i,j:entier)
variables tampon : entier

debut echange
tampon <- t[i]
t[i]   <- t[j]
t[j]   <- tampon
fin echange


programme trieTableau
variables t : tableau
          i, maxSelect, posSelect : entier

debut trieTableau
  initAleatoire t

  pour i de dim a 1 faire
    maxSelect <- max(t,1,i)
    posSelect <- position(t,maxSelect)
    echange(t,posSelect,i)
  fin pour

  affiche t
fin trieTableau
```

Corrigé exercice 3 : Cette correction ne comporte pas les commentaires qui sont bien entendu necessaires devant chaque declaration de fonction ou procedure pour rappeler ce qu'elle est cense faire. En pseudo :

```
constante dim = 7
type plateau  = tableau de dim entiers


procedure init(reference p : plateau)
variables i : entier

debut init
   pour i de 1 a dim faire
    p[i] <- 0
   fin pour
fin init


procedure affiche(reference p : plateau)
variable i : entier

debut affiche
   pour i de 1 a dim faire
     si (p[i]==0) affiche ``pile\t''
     sinon        affiche ``face\t''
     fin si
   fin pour
   affiche endl
fin affiche


fonction gagne(reference p : plateau) : entier
variable i   : entier
         rep : booleen

debut gagne
   rep = vrai
   i=1
   tant que (rep et i <=dim) faire
     rep <- (rep et p[i]=1)
   fin tantque

   si rep alors retourner 1
   sinon        retourner 0
   finsi
fin gagne


fonction coupValide(i,j : entier) : entier
variable reponse : entier

debut coupValide
  si (1<=i<=dim et 1<=j<=dim et i<> j)
     alors reponse <-1
     sinon reponse <-0
  finsi

  retourner reponse
fin coupValide
```

```
procedure flip(reference p : plateau, i,j : entier)

debut flip
  si coupValide(i,j) faire
     p[i] <- 1-p[i]
     p[j] <- 1-p[j]
  finsi

fin flip


programme jeuComplet
variables p : plateau
          i,j : entier

debut jeuComplet

init(p)
afficher ``la position de depart est \n''
affiche p

tant que (gagne(p)=0) faire
   afficher `` donner votre prochain coup ''
   lire i,j
   flip(p,i,j)
   afficher ``ce qui donne \n''
   affiche p
fin tantque

afficher ``tricheur''

fin jeuComplet
```