Les Tableaux

Un Tableau se présente sous la forme d'une suite de taille fixe d'élements de même type, et d'un ensemble d'indices qui permet de les nommer.

Interêt :
possibilité de manipuler un grand nombre de variables de même type
Accès direct à leur valeur.

Exemple :

```
type TabE = tableau d'entiers
            indexé par [Pierre,Paul,Jacques]

Variable age : TabE initialisé à {25,30,51}
```

Représentation :

age :

25	30	51
Pierre	Paul	Jacques

On accède à la valeur mémorisée à l'indice Pierre du tableau age par la notation

```
age[Pierre]
```

L'accès dans un tableau par un indice non valide provoque une erreur :

```
age[Capitaine] // erreur
```


Exemple

Analyse de températures

Un tableau compte le nombre de jours dans l'année qui ont eu à midi une température donnée.

On calcule ici la température moyenne de l'echantillon.

```
Constantes Entières Tmin  = -20
                    Tmax  = 40
Type Echantillon = tableau d'entiers indexé par [Tmin..Tmax]

Programme Moyenne
  Variables mesures  : Echantillon initialisé aléatoirement
            t        : Entier
            somme,nombre   : Entiers initialisés à zéro
  Début Moyenne
  pour t <- Tmin jusqu'à Tmax faire
    afficher "il y a eu " mesures[t]
             "jours à la température " t
    somme  <- somme  + t*mesures[t]
    nombre <- nombre + mesures[t]
  fin pour
  afficher "Sur les " nombre "jours de mesure"
           "la moyenne est de" somme/nombre
Fin Moyenne
```


Syntaxe

En pseudo langage la construction est :

```
type nom = tableau de nom_de_type
                    indexé par Indices
```

Les indices peuvent être :

- enumérés : [Pierre,Paul,Jacques]
- donnés par un intervalle : [-5..5] ['a'..'z']

Manipulation

```
Constantes Entières Min = -1
                    Max =  3
Type TabE = tableau d'entiers indexé par [Min..Max]
Programme Manipulation
 Variables t : TabE   initialisé à {8,3,-1,2,5}
           i : entier initialisé à 3
 Début Manipulation
 t[3]     <- 4
 t[i]     <- t[i]+1
 t[i-1]   <- t[(i mod 2)*2]
 t[i+1]   <- t[t[i]+i]
Fin Manipulation
```

Attention aux erreurs hors bornes !

Trouvez les ici.

Lire - Afficher

Pour lire ou afficher un tableau **t** indexé par un intervalle, on utilisera commodément les boucles **pour**

Pour le tableau précédent :

lecture

```
pour i ← Min jusqu'à Max faire
    afficher "entrer l'élément d'indice " i
    lire t[i]
fin pour
```

affichage

```
pour i ← Min jusqu'à Max faire
    afficher t[i]
fin pour
```

On a une construction du même type pour les tableaux indexé par `['a'..'z']`

On pourra aussi écrire :

```
pour i ← Pierre à Jacques dans [Pierre,Paul,Jacques] faire
   afficher t[i]
fin pour
```

En pseudo on écrira simplement

```
lire t
afficher t
```

Cryptage

Le programme suivant transforme une phrase en remplacant chaque lettre par une lettre qui lui correspond, un code.

```
Constante Entière Max  = 100
Type TabCode   = tableau de caractères indexé par ['a'..'z']
     TabPhrase = tableau de caractères indexé par [0..Max]
Programme Cryptage
  Variables code   : TabCode
            phrase : TabPhrase
            i      : Entier initialisé à zéro
            j      : Entier
            c      : Caractère
  Début Cryptage
  lire code
  afficher "entrer la phrase à coder"
  lire c
  tant que (c dans ['a'..'z'] et i <=Max) faire
    phrase[i] <- c
    i <- i+1
    lire c
  fin tant que
  afficher "le codage est "
  pour j<- 0 jusqu'à i-1 faire
   afficher code[phrase[j]]
  fin pour
  afficher "si vous voulez décoder, le code est "
  pour c <- 'a' jusqu'à 'z' faire
   afficher code[c] "->" c
  fin pour
Fin Cryptage
```


Exercices – Cours

1) **(Scalaire)** Ecrire un algorithme qui calcule le produit scalaire de deux tableaux a et b indexés par $[-3..5]$

2) **(Miroir)** Ecrire un programme qui remplit un tableau b avec les éléments d'un tableau a repris en ordre inverse. Le tableau a pourra être initialisé aléatoirement.

Corrigé

1) **(Scalaire)**

```
Constantes Entières A = -3
                    B = 5  // bornes du tableau
Type vecteur = tableau d'entiers indexé par [A..B]

Programme Scalaire
 Variables
    a,b  : vecteurs initialisés aléatoirement
    i,sc : entiers
 Début scalaire
  // au depart sc est nul
  sc <- 0
  // calcul itératif du produit
  pour i <- A jusqu'à B faire
      sc <- sc + a[i]*b[i]
  fin pour
  // affichage du resultat
  afficher "Le produit scalaire de" a
           "par" b "vaut" sc
 Fin scalaire
```

Invariant :

$$sc = \sum_{k=A}^{i-1} a_k * b_k \text{ et } i \in [A..B+1]$$

2) **(Miroir)**

```
Constante Entière B=99
Type TabE = tableau d' entiers indexé par [0..B]

Programme Miroir
 Variables    a,b  : TabE
              i    : entier
 Debut Miroir
  initialisation aléatoire de a
  pour i <- 0 à B faire
      b[i] <- a[B-i]
  fin pour

  afficher "le tableau " a
           "a pour miroir" b
fin Miroir
```

Invariant :

$$miroir(a[B+1-i..B]) = b[0..i-1] \text{ et } i \in [0..B+1]$$

Dans le cas limite où $i = 0$, on considère que $a[B+1..B]$ et $b[0..-1]$ sont des mots vides, on prend alors pour convention qu'ils sont miroirs l'un de l'autre.

Le cas du C++

En C++, la définition de tableaux ne peut se faire que pour des index qui sont des intervalles de la forme $[0..N]$.

Le passage du pseudo au C++ demandera une transformation.

Syntaxe

```
const int dim =100;
typedef int TabE[dim]; //declaration d'un type TabE
void main() {   // debut du programme
TabE t={2,5,4}; // declaration de variable
}
```

A remarquer : l'initialisation partielle. C++ complète alors les autres cases du tableau par des zéros.

Exemple-Températures

Ici, il faut de plus pouvoir initialiser aléatoirement un tableau.

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
const int Tmin  = -20;
const int Tmax  =  40;
typedef int Echantillon [Tmax-Tmin+1];  // dimension
void main(){                      // Programme Moyenne
  Echantillon mesures;
  int t,somme=0,nombre=0;
  // initialisation aléatoire
  srand(time(NULL));
  for (t=Tmin; t<=Tmax; t++){
    mesures[t-Tmin] = rand() % 50; // decalage
  }
  for (t=Tmin; t<=Tmax; t++){
    cout << "il y a eu " << mesures[t-Tmin] // decalage
         << " jours à la température " << t;
    somme  = somme  + t*mesures[t-Tmin];    // decalage
    nombre = nombre + mesures[t-Tmin];      // decalage
  } // fin pour
  cout << "Sur les " << nombre << " jours de mesure "
       << " la moyenne est de "  << somme/nombre << endl;
}
```


Exemple Cryptage

```
#include<iostream.h>
const int Max  = 100;
typedef char TabCode['z'-'a'+1]; // de 'a' à 'z'
typedef char TabPhrase[Max+1];   // de 0 a Max

void main(){
 TabCode code;
 TabPhrase phrase;
 int i=0,j;
 char c;
 for(c='a';c<='z';c++){
   cout << "entrer le code pour " << c << " : ";
   cin >> code[c-'a']; // decalage
 }  // fin for
 cout << "entrer la phrase à coder ";
 cin >> c;
 while ((('a' <= c) && (c <='z')) && i<=Max){
   phrase[i]= c;
   i++;
   cin >>c;
 }  // fin while
 cout << "le codage est " << endl;
 for (j=0;j<=i-1;j++){
   cout << code[phrase[j]-'a']; //decalage
 }  // fin for
 cout << endl << "si vous voulez décoder : ";
 for (c='a'; c<='z'; c++){
   cout <<  code[c-'a'] << " -> " c << endl ; //decalage
 }  // fin for
}
```


Exercices – Cours

Traduire en C++ les exercices précédents

Corrigé

1) **(Scalaire)**

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
const int A = -3, B = 5;      // les bornes du tableau
typedef int vecteur[B-A+1];

void main(){ // Programme Scalaire
  vecteur a,b;
  int     i,sc;
  // initialisation aleatoire de a et b avec des chiffres
  srand(time(NULL));
  for (i=A;i<=B;i++){
   a[i-A]=rand() % 10;
   b[i-A]=rand() % 10;
  }   // fin pour

  cout << "Le produit scalaire des vecteurs " << endl;
  for (i=A;i<=B;i++){ cout << a[i-A] << ' ';
  }   // fin pour
  cout << endl << " et " << endl;
  for (i=A;i<=B;i++){ cout << b[i-A] << ' ';
  }   // fin pour
  cout << endl << "vaut ...";

  sc = 0;
  for (i=A;i<=B;i++){
      sc = sc + a[i-A]*b[i-A];
  }   // fin pour
  cout << sc << endl;
}
```


2) **(Miroir)** Dans le cas ou la borne inférieure des index est 0, on obtient un code C++ plus proche du pseudo :

```
#include<iostream>
#include <stdlib.h>
#include <time.h>
const int B=99;
typedef int TabE[B+1]; // attention au +1 !!

void main(){
 TabE a,b;
 int i;

 // initialisation aléatoire de a
 srand(time(NULL));
 for (i=0;i<=B;i++){
   a[i]=rand() % 10;
 }

 for (i=0;i<=B;i++){
   b[i] = a[B-i];
 } // fin pour

 cout << "le tableau ";
 for (i=0;i<=B;i++){
   cout << a[i] << ' ';
 }
 cout << endl << "a pour miroir ";
 for (i=0;i<=B;i++){
   cout << b[i] << ' ';
 }
}
```


Exercice – Cours
Recherche d'un élément

Ecrire un algorithme qui indique si un élément donné est dans un tableau.

Corrigé
Recherche d'un élément

```
Contante A = 5
         B = 10
Type TabE = tableau d'entiers indexé par [A..B]

Programme Recherche
 Variables t         : TabE initialisé aléatoirement
           i,x       : entiers
           trouvé : booléen
 Début Recherche
 affiche "entrer l'élément recherché "
 lire x
 pour i <- A jusqu'à B faire
   si (t[i]=x) alors trouvé <- vrai
   fin si
 fin pour
 affiche "Le Pb est de savoir si " x " est dans "
 affiche t
 affiche  "La réponse est  " trouvé
fin Recherche
```

Invariant :

$$A \leq i \leq B+1 \text{ et trouvé} = (x \in t[A..i-1])$$

Exercice

Que ce passerait-il si on avait écrit les choses suivantes ?
Compléter éventuellement les ...

```
i<-A
tant que (t[i] <> x) faire i <- i+1
fin tant que

i<-A
tant que (t[i] <> x et i<= B) faire i <- i+1
fin tant que

i<-A
tant que (t[i] <> x et i< B) faire i <- i+1
fin tant que
si .... alors afficher "présent"
sinon afficher  "pas présent"
fin si

t[B] <- x
i<-A
tant que (t[i] <> x) faire i <- i+1
fin tant que
si .... alors afficher "présent"
sinon afficher  "pas présent"
fin si
```


Corrigé – Recherche

```
i<-A
tant que (t[i] <> x) faire i <- i+1
fin tant que
```

Débordement et pas d'arrêt

```
i<-A
tant que (t[i] <> x et i<= B) faire i <- i+1
fin tant que
```

Débordement si $x \notin t$: lorsque i passe à $B+1$, l'évaluation de la condition passe par un accès hors bornes.

Corrigé – Recherche

```
i<-A
tant que (t[i] <> x et i< B) faire i <- i+1
fin tant que
si t[i]=x alors afficher "présent"
sinon afficher  "pas présent"
fin si
```

Invariant : $i \in [A..B]$ et $x \notin [A..i-1]$
Terminaison : i croit strictement. Arrêt si $i \geq B$

```
t[B] <- x // On ecrase la valeur ici !
i<-A
tant que (t[i] <> x) faire i <- i+1
fin tant que
si i<B alors afficher "présent"
sinon aficher  "pas présent"
fin si
```

Terminaison : i croit strictement partant de A avec un pas de 1, on atteindra donc $i = B$ pour lequel la condition est fausse.

même invariant

A la sortie :
$i \in [A..B]$ et $t[i] = x$ et $x \notin [A..i-1]$

Recherche Dichotomique

Si l'on suppose que le tableau est trié,
on peut accélérer la recherche

```
Contantes A = 5
          B = 10
Type Tentier = tableau d'entiers indexé par [A..B]
Programme RechercheTrie
 Variables t            : Tentier
           x,milieu     : entiers
           gauche       : entier initialisé à A
           droite       : entier initialisé à B
 Debut RechercheTrie
 affiche "entrez un tableau trié "       lire t
 affiche "entrez l'élément recherché "   lire x
 tant que (gauche < droite ) faire
   milieu <- (gauche+droite) div 2
   si (t[milieu] < x)
     alors gauche <- milieu +1
   sinon
     droite <- milieu
   fin si
 fin tant que
 affiche "Le Pb est de savoir si " x " est dans "
 affiche t
 affiche  "La réponse est  "
 si (t[gauche]= x) alors afficher " oui "
 sinon                   afficher " non "
 fin si
fin RechercheTrie
```

Terminaison :

|droite − gauche| est décroissant strictement jursqu'à 0

car si $g < d$ alors $g \leq m < d$ et, soit g croit strictement, soit d décroit strictement

Invariants :

t trié et gauche $\leq$ droite et
($x \in t$[gauche..droite] ou $x \notin t$)

Correction :

Au final on a $\neg$(gauche $<$ droite) et gauche $\leq$ droite, et

$$x \in t[\texttt{gauche}..\texttt{droite}] \text{ ou } x \notin t$$

Donc

$$x = t[\texttt{gauche}] \text{ ou } x \notin t$$

Exercices – TD

3) **(Echange)** Écrire un programme qui échange le plus grand élément d'un tableau avec son dernier élément.

4) **(Inverse)** Écrire un programme qui modifie un tableau en mettant ses éléments en ordre inverse. Pour ce problème on impose que votre programme n'utilise qu'un seul tableau.

5) **(Echantillonage)**

Écrire un programme qui demande à l'utilisateur d'entrer une suite de lettres minuscules. Le programme maintient à jour un tableau dont les cases sont indexées par des caractères minuscules, qui maintient le compte du nombre des occurences de chacunes des lettres.

Afficher ensuite le tableau sous la forme :

```
a | *
b | *****
⋮ | ⋮
z | ******
```

avec une $*$ par occurence de lettres

L'exemple précédent peut correspondre à l'entrée "zzbzbazbbbzz"

6) **(Polynome)**

Un tableau peut servir à représenter un polynome :

à $Q(x) = a_0.x^n + a_1.x^{n-1} + \ldots + a_{n-1}.x + a_n$,
on associe le tableau indexé par $[0..n]$:

$$[a_0, .., a_n]$$

Que fait le programme suivant ?

```
Constante Entière N = 4
Type Polynome = tableau de réels indexé par [0..N]
Programme Mystère
  Variables p : Polynome initialisé à {1,0,2,3,-1}
            i : entier
            v,x : réels
  Debut Mystère
  afficher "entrez une valeur pour x"
  lire x
  v <- 0
  pour i <- 0 à N faire
    v <- v*x + p[i]
  fin pour

  afficher  "le resultat est " v
fin Mystère
```

Il faut aussi le prouver !

Corrigés

3) **(Echange)**

```
Constantes Entière A=-3
                   B=5 // les bornes du tableau
Type   TabE = tableau d'entiers indexé par [A..B]

Programme EchangeMaxDernier
Variables
    a              : TabE initialisé aléatoirement
    pos,max,i      : entiers

Debut EchangeMaxDernier
  afficher "initialement on a" a
  // on commence par un maximum provisoire
  pos <- A
  max <- a[A]
  pour i <- A+1 a B faire
    si (a[i] > max) alors
       max <-  a[i]
       pos <- i
    fin si
  fin pour
  // affichage du résultat intermédiaire
  afficher "le max est" max "par exemple en position" pos
  // Echange des cases
  a[pos]      <- a[B]
  a[B]        <- max
fin EchangeMaxDernier
```

Invariant :

$i \in [A..B+1]$ et max $= a[$pos$]$ et max $= sup\{t[A..i-1]\}$

```
#include<iostream.h>
#include <stdlib.h>
#include <time.h>
const int A=-3;
const int B=5;          // les bornes du tableau
typedef int TabE[B-A+1];
void main(){ //Programme EchangeMaxDernier
  TabE a;
  int pos,max,i;
  // initialisation aléatoire de a
  srand(time(NULL));
  for(i=A;i<=B;i++) {a[i-A]=rand()%50;
  }
  cout << "initialement on a ";
  for(i=A;i<=B;i++) {cout << a[i-A]<<' ';
  }
  cout << endl;
  // on commence par un maximum provisoire
  pos = A;
  max = a[A];
  for(i=A+1;i<=B;i++){
    if (a[i-A] > max){
      max =  a[i-A];
      pos = i;
    }
  }
  // affichage du résultat intermédiaire
  cout << "un max est " << max << " en position" << pos;
  // Echange des cases
  a[pos-A] = a[B-A];
  a[B-A]   = max;
}
```


4) (Inverse II)

```
Constantes Entières A = -5
                    B =  3
Type TabE = tableau d'entiers indexé par [A..B]

Programme InverseII
 Variables   a     : TabE initialisé aléatoirement
             aux,i : entiers
 Debut InverseII
  afficher "initialement on a" a
  pour i <- A jusqu'à (A+B) div 2 faire
      aux  <- a[i]         // sauvegarde
      a[i] <- a[B+A-i]    // puis
      a[B+A-i] <- aux    // échange
  fin pour   // affichage du resultat
  afficher "et pour miroir" a
fin InverseII
```

Invariant :

$i \in [A..(A+B)\ div2\ +1]$
et $a[A..i-1] = \texttt{miroir}(a_0[B+A-i+1..B])$
et $a[B+A-i+1..B] = \texttt{miroir}(a_0[A..i-1])$


```
#include<iostream.h>
#include <stdlib.h>
#include <time.h>
const int A=-3;
const int B=5;          // les bornes du tableau
typedef int TabE[B-A+1];

void main(){ //Programme Inverse
  TabE a;
  int aux,i;
  // initialisation aléatoire de a
  srand(time(NULL));
  for(i=A;i<=B;i++) {a[i-A]=rand()%10;
  }

  cout << "initialement on a ";
  for(i=A;i<=B;i++) {cout << a[i-A]<<' ';
  }
  cout << endl;
  for(i=A; i<= (A+B)/ 2;i++){
    aux= a[i-A];      // sauvegarde
    a[i-A]= a[B-i];  // puis
    a[B-i] = aux;   // échange
  } // fin for
  cout <<  "et pour miroir";

  for(i=A;i<=B;i++){
    cout <<a[i-A] << ' ';
  }
}
```


5) (Echantillonage)

```
Type vecteur = tableau d'entiers indexé par ['a'..'z']

Programme Echantillon
 Variables
    a    : vecteur initialisé à 0
    c    : caractère
    i    : entier
 Debut Echantillon
  // Entrée des caractères
  afficher "Entrez une suite de caractères"
  lire c
  tant que c dans ['a'..'z'] faire
    a[c] <- a[c] +1
    lire c
  fin tant que

  // présentation du résultat
  pour c <- 'a' jusqu'à 'z' faire
    afficher c "\t : "
    pour i <- 1 jusqu'à a[c] faire
      afficher "*"
    fin pour
    passer à la ligne
  fin pour
 Fin Echantillon
```

```
#include<iostream.h>
typedef int vecteur['z'-'a'+1];

void main(){
 vecteur a={0}; // Rq l'initialisation à 0
 char c;
 int i;

 cout << "Entrez une suite de caractères";
 cin >> c;
 while( (c>='a')&& (c<='z')){
   a[c-'a'] =a[c-'a'] +1;
   cin >>c;
 }

 for(c='a';c<='z';c++){
   cout << c << "\t : ";
   for(i=1; i<= a[c-'a'];i++){
     cout << "*";
   }
   cout << endl;
 }
}
```


6) **(Polynome)**

Invariant :

$$v = \sum_{k=0}^{i-1} p[k] * x^{i-1-k} \texttt{ et } i \in [0..N+1]$$

Au final, lorsque $i = N + 1$ on a :

$$v = \sum_{k=0}^{N} p[k] * x^{N-k} = P(x)$$

On a utilisé la décomposition :

$$P(x) = (\ldots((a_0 * x + a_1) * x + a_2) * x + \ldots) * x + a_n$$

L'intérêt étant ici d'éviter de faire des multiplications.

(N multiplications et N additions)