TD-1 Rappels
Enumerations, Structures, Tableaux, Chaines de Caractères

Année 2002

Les Énumérations

En $C++$, on peut enrichir les types de base (entiers, réels, caractères ...) en définissant des types composés. Une première possibilité est d'utiliser l'énumération qui introduit des constantes particulières.

```
#include<iostream>
typedef enum mouvement {monte,descend};
int main(){
int i;
mouvement cabine_mvt;
if (cabine_mvt==monte) cout << ``elle monte''
   else cout << ``elle descend''
}
```

`monte` et `descend` sont alors des constantes du langage et sont des entités de type `mouvement`.

Les Structures

Une seconde extension des types de base est construite sur les structures, par exemple, pour définir dans le programme principal une cabine c de couleur jaune et qui monte, on pourra écrire:

```
#include<iostream>
typedef enum mouvement {monte, descend};
typedef enum couleur {bleu,jaune,rouge};

typedef struct cabine {
  mouvement sens;
  couleur aspect;
};
int main(){
 cabine c;
 c.sens=monte;
 c.aspect=jaune;
}
```

Les Tableaux

Ils permettent de regrouper en une seule entité des données de type identique, y compris des structures ou des énumérations.

```
#include<iostream>
#include<cstdlib>
#include<ctime>

const int dim = 10;
typedef enum couleur {bleu,jaune,rouge};
```

```
typedef couleur flash[dim];

int main(){
 flash t;
 int i;

 srand(time(NULL));

 for(i=0;i<dim;i++) {
   switch (rand() % 3){
     case 0 : flash[i]=bleu;break;
     case 1 : flash[i]=jaune;break;
     default: flash[i]=rouge;
   }
 }
}
```

Remarquez que la numérotation des cases du tableau va de 0 à 9, (ce qui fait 10 élements). Lorsque les tableaux sont passés en paramètre à une fonction ou une procédure, ils le sont toujours par référence (on en verra l'explication dans quelques cours avec les pointeurs).

Compléments : Pour générer un entier aléatoirement on peut utiliser la fonction **rand()**. Pour restreindre cet entier par exemple dans l'intervalle [0..9], il suffit de prendre son reste modulo 10 : $x = 132 \ \% \ 10$ est équivalent à $x = 2$.

Pour que le générateur de nombres aléatoires donne des résultats différents pour deux exécutions différentes, il faut être capable de l'initialiser différement entre deux exécutions. Pour cela une possibilité est de l'initialiser en fonction de l'heure courante. Cela peut être fait au debut du programme principal en invoquant la procédure **srand(time(NULL))**. **rand** et **srand** sont définies dans le fichier entête **cstdlib**, et **time** dans le fichier entête **ctime** qu'il ne faut pas oublier d'inclure au debut du programme.

Exercice 1 (TD) : Définir un type carte bleue qui contienne un champ **montant** qui modélise l'argent disponible, un champ **code** qui contienne le code secret, un champ **type_carte** qui sera soit visa, soit mastercard.

Exercice 2 (TD) : Donner une structure de données qui permette de représenter un jeu de 32 cartes complet.

Exercice 3 (TD-TP) : Ecrire une fonction qui donne la valeur d'une carte à la belote. (Attention il y a un atout). La correspondance des valeurs est donnée par le tableau :

carte	7	8	9	10	valet	dame	roi	as
standard	0	0	0	10	2	3	4	11
atout	0	0	14	10	20	3	4	11

Les Chaines de caractères

Une chaine de caractère est constituée d'une suite de caractères. Ceux–ci sont représentés en interne par des codes définis par la norme ASCII (*American Standard for Information Interchange*). A titre d'information on a la correspondance :

```
  0 NUL   1 SOH   2 STX   3 ETX   4 EOT   5 ENQ   6 ACK   7 BEL
  8 BS    9 HT   10 NL   11 VT   12 NP   13 CR   14 SO   15 SI
 16 DLE  17 DC1  18 DC2  19 DC3  20 DC4  21 NAK  22 SYN  23 ETB
 24 CAN  25 EM   26 SUB  27 ESC  28 FS   29 GS   30 RS   31 US
 32 SP   33 !    34 "    35 #    36 $    37 %    38 &    39 '
 40 (    41 )    42 *    43 +    44 ,    45 -    46 .    47 /
 48 0    49 1    50 2    51 3    52 4    53 5    54 6    55 7
 56 8    57 9    58 :    59 ;    60 <    61 =    62 >    63 ?
 64 @    65 A    66 B    67 C    68 D    69 E    70 F    71 G
 72 H    73 I    74 J    75 K    76 L    77 M    78 N    79 O
 80 P    81 Q    82 R    83 S    84 T    85 U    86 V    87 W
 88 X    89 Y    90 Z    91 [    92 \    93 ]    94 ^    95 _
 96 `    97 a    98 b    99 c   100 d   101 e   102 f   103 g
104 h   105 i   106 j   107 k   108 l   109 m   110 n   111 o
112 p   113 q   114 r   115 s   116 t   117 u   118 v   119 w
120 x   121 y   122 z   123 {   124 |   125 }   126 ~   127 DEL
```

Par exemple le code de 'A' est 65. Cela explique que l'on puisse faire sur les caractère certaines opérations comme on le fait pour les entiers : 'A' + 42 c'est 'a', ou on peut tester si 'A' $\leq$ 'a'.

Le type caractère en C est traduit par `char`, le type chaîne de caractère est `string`. Ce dernier est défini dans le fichier entête `<string>`

Par exemple `"une chaîne"` est une chaîne de type `string`. Les chaînes sont munies des opérations suivantes :

- la concaténation : `"c" + s1` qui retourne une chaîne
- `s.length()` qui retourne la longueur de `s`
- le test du vide de la chaîne est fait par : `s.empty()`
- le n-ème caractère est donné par : `s[n-1]`.
- pour extraire une sous-chaîne : `s.substr(position,longueur)`
- insérer une chaîne : `s.insert(position,s1)`

Attention, comme pour les tableaux, les positions utilisées dans une chaîne de caractères ne doivent pas dépasser la longueur de la chaîne ! Par contre il est à noter que le passage des `string` en paramètre est libre soit par `valeur` soit par `référence`, il ne s'agit donc pas d'un tableau de caractères que l'on aurait maquillé !

On peut remarquer la façon dont on invoque les "fonctions d'un objet" de type `string`.

Exercice 4 (TD) : Exécuter à la main les lignes suivantes

```
string s;
bool b;
int k;
...
s = "Bonjour";
s = s + " tout le monde !";
cout << s << " contient " << s.length() << " caracteres"
b = s.empty();
cout << s[4];
cout << s.substr(3,6);
s.insert(1,"- ");
```

Exercice 5 (TP) : Ecrire une fonction qui prend comme paramètre un caractère et renvoie le caractère majuscule s'il s'agit d'une lettre ou le caractère initial sinon. Ecrire une procédure qui transforme, dans une chaîne, tout caractère minuscule en majuscule.

Exercice 6 (TP) : Ecrire un programme qui transforme une chaîne de caractères en un palindrome.

Par exemple : `taratata` sera transformée en `taratataatatarat`

Exercices complémentaires

Exercice 7 : On peut remarquer que le plus petit palindrome qui étend `abb` est `abba`. On veut écrire un programme qui transforme une chaîne de caractères en plus petit palindrome qui l'étend. Pour cela :

- écrire une fonction `bool est_palindrome(string s)`
- écrire une fonction `int position_plus_grand_suffixe_palindrome(string s)`
- Utiliser les fonctions précédente pour résoudre le problème.

Exercice 8 : Ecrivez une procédure qui mélange un jeu de 32 cartes tel qu'on l'aura défini à l'exercice 2.

Exercice 9 : Supposons que le code d'une carte bleue soit choisi au hasard lors de sa création, écrire **initialise_carte(... ma_carte, ... depot_initial, ... type_choisi)**, qui retournera le code.

Exercice 10 :

a) Ecrire une procédure $triangle(a, b)$ qui, supposant que le tableau a soit une ligne du triangle de Pascal, calcule la ligne suivante en la plaçant dans b.

b) Ecrire une procédure $permute\_g(a)$ qui permute circulairement les éléments d'un tableau a, en les déplaçant d'une case vers la gauche, le premier élément devenant le dernier.

c) Idem avec $permute\_d$ qui permute les élément du tableau de la même façon, mais vers la droite.

d) Ecrire une procédure $triangle\_direct(a)$ qui calcule la ligne suivante du triangle de Pascal, sans l'aide de tableaux auxilliaires.

e) Soient deux tableaux a et b de taille n dont les élélements sont déjà triés et un tableau c de taille $2n$ vide. Ecrire une fonction $fusion(a, b, c)$ qui fusionne les éléments des tableaux a et b dans le tableau c de telle sorte que les éléments du tableau c soient eux aussi triés.

Correction

<u>Consignes</u> Essayez de traiter les exercices 5 et 6 en TP. Par contre faites bien les 4 premiers en TD.

Exercice 1 : A noter le choix qui est fait d'avoir un code stocké dans un tableau d'entier. On pourra indiquer les conséquences que ça a : il faudra s'assurer que l'on a bien des chiffres via les fonctions de définition d'une carte.

```
#include <iostream>

typedef enum geree_par {visa,mastercard};

typedef struct carte_bleue {
  int montant;
  int code[4];
  geree_par type_carte;
};

int main(){
  carte_bleue c;

  c.montant = 100;
  c.code[0]=1
  c.code[1]=4
  c.code[2]=1
  c.code[3]=8;
  c.type_carte=visa;
}
```

Exercice 2 : Pour le TP, je deconseillerai de commencer par celui là qui est un peu long à taper, autant faire les autres d'abord. Pour les mêmes raisons on n'est pas obligé de donner la correction complète en TD.

A remarquer : on utilise les switch-case pour décrire les énumérations. Ne pas oublier le break !

Pour définir toutes les cartes j'ai construit des tableaux de correspondance qui associent un numéro à chaque constantes d'un type énuméré.

Il est intéressant de souligner que l'on peut utiliser le même nom `affiche` pour différents types, sans qu'il y ait ambiguité.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

typedef enum couleur {coeur,trefle,pique,carreau};
typedef enum valeur {as,roi,dame,valet,dix,neuf,huit,sept};

typedef struct carte {
  couleur c;
  valeur v;
};

typedef carte jeu_c[32];

void affiche(couleur c){
 switch(c) {
   case coeur   : cout << "coeur";   break;
   case trefle  : cout << "trefle";  break;
   case pique   : cout << "pique";   break;
   case carreau : cout << "carreau"; break;
   }
}

void affiche(valeur v){
 switch(v) {
   case as    : cout << "as";    break;
```

```
  case roi   : cout << "roi";   break;
  case dame  : cout << "dame";  break;
  case valet : cout << "valet"; break;
  case dix   : cout << 10 ;     break;
  case neuf  : cout << 9;       break;
  case huit  : cout << 8;       break;
  case sept  : cout << 7;       break;
  }
}

void affiche(carte c){
 affiche(c.v);
 cout <<" de ";
 affiche(c.c);
 cout << endl;
}

void affiche(jeu_c j){
 int i;
 for (i=0;i<32;i++){
   affiche(j[i]);
 }
}

int main(){
  int i,j,n=0;
  jeu_c jeu;

  couleur corresp_couleur[4]={coeur,trefle,pique,carreau};
  valeur corresp_valeur[8]={as,roi,dame,valet,dix,neuf,huit,sept};

  for (i=0;i<8;i++){
   for (j=0;j<4;j++){
    jeu[n].c=corresp_couleur[j];
    jeu[n].v=corresp_valeur[i];
    n++;
   }
  }
 affiche(jeu);
}
```

Exercice 3 : Il faudra discuter avec les étudiants de la façon d'introduire l'atout dans le contexte de la fonction.

```
#include <iostream>

typedef enum couleur {coeur,trefle,pique,carreau};
typedef enum valeur {as,roi,dame,valet,dix,neuf,huit,sept};

typedef struct carte {
  couleur c;
  valeur v;
};

int valeur(carte c,couleur atout){
  switch (c.v) {
  case as : return 11;
  case dix : return 10;
  case roi : return 4;
  case dame : return 3;
  case valet : {
    if (c.c==atout)
      return 20;
    else return 2;
```

```
  }
  case neuf : {
    if (c.c==atout) return 14;
    else return 0;
  }
  default : return 0;
  }
}

int main(){
  carte x;

  x.v =neuf;
  x.c=coeur;

  cout << valeur(x,coeur) << " " << valeur(x,pique);
}
```

Exercice 4 : Il y a 23 caractères avec les espaces, `b` recoit `faux`, `s[4]` est 'o' puisqu'on commence à zéro, "jour t" est la sous chaîne, et le resultat de l'insertion est "B- onjour tout le monde !"

Exercice 5 : Il faut remarquer les comparaisons directes entre caractères via leur code ASCII, ce qui permet le changement d'origine.

Il ne faut pas oublier le passage en référence de la chaine, qui n'est décidément pas un tableau. Ici encore j'ai donné le même nom `minToMaj` aux 2 fonctions.

```
#include<iostream>
#include<string>

char minToMaj(char c){
 if (('a'<= c) && (c<='z')) return (c-'a'+'A');
 else return c;
}

void minToMaj(string &s){
 int i;
 for (i=0;i<s.length();i++){
 s[i]= minToMaj(s[i]);
 }
}

int main(){
 string s;
 cin >> s;
 minToMaj(s);
 cout << s;
}
```

Exercice 6 : Ici encore je l'ai fait avec une boucle en sens inverse. On peut poser la question en prenant `i` croissant.

```
#include <string>
#include<iostream>

int main(){
   int i;
   string s,pal;
   cin >> s;
   pal=s; // rq c'est une copie
   for (i=s.length()-1;i>=0;i--){
      pal=pal+s[i];
   }
   cout << pal << endl;
   cout << s << endl;
}
```

Exercice 7 :
```
#include <iostream>
#include <string>

bool est_palindrome(string s){
   int i;
   bool res=true;
   int L=s.length();
   for(i=0;i<=L/2-1;i++){  // Rq en exercice on peut donner une solution avec un while
      res=res && (s[i]==s[L-1-i]);
   }
   return res;
}

int position_plus_grand_suffixe_palindrome(string s){
   int i;
   string s1;
   i=0;
   s1=s;
   while ( i < s.length() && !(est_palindrome(s1)) ){
      i++;
      s1=s.substr(i,s.length()-i);
   }
   return i;
}

void donnePalindrome(string &s){
   int i=position_plus_grand_suffixe_palindrome(s);
   //qq cas particulier
   if (i!=s.length()-1 | s.length()==1) i--;
   for (;i>=0;i--){
      s+=s[i];
   }
}

int main(){
   string c;
   while(true){
      cin >> c;
      cout << "qui est un palindrome :" << est_palindrome(c) <<endl;
      donnePalindrome(c);
      cout << c << endl;
   }
}
```

Exercice 8 : Pour mélanger, j'ai choisi d'effectuer un nombre aléatoire de permutations entre 2 cartes du jeu. On peut imaginer d'autres façon de mélanger, par exemple on coupe le paquet et on intercale les cartes. C'est peut être une source d'exercices originaux pour IF121 (on passe de trier à mélanger :o)).

```
// echange les cartes de j de positions pos1 et pos2
// elles doivent etre entre 0 et 31 !
void echange(jeu_c j, int pos1, int pos2){
 carte aux;
 aux=j[pos1];
 j[pos1]=j[pos2];
 j[pos2]=aux;
}

// melange le jeu en effectuant moins de mille echange de cartes 2 a 2
void melange(jeu_c j){
int nb, pos1,pos2;
srand(time(NULL));

 for (nb=rand()%1000;nb >0;nb--){
```

```
  pos1=rand()%32;
  pos2=rand()%32;
  echange(j,pos1,pos2);
 }
}

.... et on ajoute ca au main()
 melange(jeu);
 cout << "une fois melange ca donne" << endl;
 affiche(jeu);
....
```

Exercice 9 : Insister sur le profil. Pour varier les plaisirs je n'ai pas tiré 4 chiffres au hazard, mais un seul nombre de 4 chiffres.

```
#include <iostream>

#include <cstdlib>
#include <ctime>

typedef enum geree_par {visa,mastercard};

typedef struct carte_bleue {
  int montant;
  int code[4];
  geree_par type_carte;
};

int initialise_carte(carte_bleue &c,int depot_initial, geree_par type_choisi){
  int code_genere;
  code_genere = rand()%10000;

  c.montant=depot_initial;
  c.type_carte=type_choisi;
  c.code[3]=code_genere%10;
  code_genere=code_genere/10;
  c.code[2]=code_genere%10;
  code_genere=code_genere/10;
  c.code[1]=code_genere%10;
  code_genere=code_genere/10;
  c.code[0]=code_genere%10;

  return code_genere;
}

int main(){
  carte_bleue c;

  srand(time(NULL));

  cout << initialise_carte(c,100,visa) << endl;
}
```

Exercice 10 a) : Le triangle de Pascal est la matrice des C_n^p. Elle se calcule de bas en haut avec la règle $C_{n+1}^p = C_n^{p-1} + C_n^p$.

```
#include <iostream>

const int dim=10;
typedef int tableau[dim];

void triangle(tableau a, tableau b){
  int i;
```

```
    b[0]=1;
    // on suppose que les entiers non significatifs sont nuls
    for(i=1;i<dim;i++){
      b[i]=a[i]+a[i-1];
    }
  }

  void affiche(tableau t){
    int i;

    for(i=0;i<dim;i++){
      cout << t[i] << " ";
    }
    cout << endl;
  }

  void init(tableau t){
    int i;
    for (i=0;i<dim;i++){
      t[i]=0;
    }
  }

  int main(){
    tableau a,b;

    init(a);
    init(b);
    a[0]=1; a[1]=2; a[2]=1;

    triangle(a,b);
    affiche(b);
  }
```

Exercice 10 b) : Attention, la dernière case du tableau a pour numéro $dim-1$ (mon bug).

```
#include <iostream>

const int dim=10;
typedef int tableau[dim];

void affiche(tableau t){
  int i;

  for(i=0;i<dim;i++){
    cout << t[i] << " ";
  }
  cout << endl;
}

void init(tableau t){
  int i;
  for (i=0;i<dim;i++){
    t[i]=0;
  }
}

void permute_g(tableau t){
  int i,aux;

  aux=t[0];
  for(i=1;i<dim;i++){
    t[i-1]=t[i];
```

```
  }
  t[dim-1]=aux;
}

int main(){
  tableau a;

  init(a);
  a[0]=1; a[1]=2; a[2]=1;

  permute_g(a);
  affiche(a);

}
```

Exercice 10 c) : Là il ne faut rien écraser, la boucle se déroule donc dans l'autre sens.

```
#include <iostream>

const int dim=10;
typedef int tableau[dim];

void affiche(tableau t){
  int i;

  for(i=0;i<dim;i++){
    cout << t[i] << " ";
  }
  cout << endl;
}

void init(tableau t){
  int i;
  for (i=0;i<dim;i++){
    t[i]=0;
  }
}

void permute_d(tableau t){
  int i,aux;

  aux=t[dim-1];
  for(i=dim-1;i>0;i--){
    t[i]=t[i-1];
  }
  t[0]=aux;
}

int main(){
  tableau a;

  init(a);
  a[0]=1; a[1]=2; a[2]=1;

  permute_d(a);
  affiche(a);
}
```

Exercice 10 d) : La seule remarque c'est que ça se fait directement, sans rien écraser si on lit de droite à gauche.

```
#include <iostream>

const int dim=10;
typedef int tableau[dim];
```

```
void affiche(tableau t){
  int i;

  for(i=0;i<dim;i++){
    cout << t[i] << " ";
  }
  cout << endl;
}

void init(tableau t){
  int i;
  for (i=0;i<dim;i++){
    t[i]=0;
  }
}

void triangle_direct(tableau t){
  int i,aux;

  for(i=dim-1;i>0;i--){
    t[i]=t[i-1]+t[i];
  }
  t[0]=1;
}

int main(){
  tableau a;

  init(a);
  a[0]=1; a[1]=2; a[2]=1;

  triangle_direct(a);
  affiche(a);

  triangle_direct(a);
  affiche(a);
}
```

Exercice 9 e) : `#include <iostream>`

```
const int dim=3;
typedef int tableau1[dim];
typedef int tableau2[dim*2];

void affiche1(tableau1 t){
  int i;

  for(i=0;i<dim;i++){
    cout << t[i] << " ";
  }
  cout << endl;
}

void affiche2(tableau2 t){
  int i;

  for(i=0;i<2*dim;i++){
    cout << t[i] << " ";
  }
  cout << endl;
}
```

```
void fusionne(tableau1 a, tableau1 b, tableau2 res){
  int i,j,k;

  j=0;
  k=0;

  for(i=0;i<2*dim;i++){
    if( (k==dim) | ( (j<dim) && (a[j]<= b[k]) ) ) {
      res[i]=a[j];
      j++;
    }
    else {
      res[i]=b[k];
      k++;
    };
  }

}

int main(){
  tableau1 a,b;
  tableau2 c;

  a[0]=1; a[1]=2; a[2]=4;
  b[0]=2; b[1]=3; b[2]=5;

  affiche1(a);
  affiche1(b);

  fusionne(a,b,c);
  affiche2(c);
}
```