TD-1
Rappels

Enumerations et Structures

En $C++$, on peut enrichir les types de bases (entiers, réels, caractères ...) en définissant des types composés. Une première possibilité est d'utiliser l'énumeration.

```
typedef enum mouvement {monte,descend};
void main(){
int i;
mouvement cabine_mvt;
}
```

Une seconde est d'utiliser les structures, par exemple, pour definir dans le programme principal une cabine c de couleur jaune et qui monte, on pourra écrire:

```
typedef enum mouvement {monte, descend};
typedef enum couleur {bleu,jaune,rouge};

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

Exercice 1 (TD-TP) :

Donner une structure de données qui permette de représenter un jeu de 32 cartes.

Exercice 2 (TD) :

Définir un type carte bleue qui contienne un champs **montant** qui modélise l'argent disponible, un champs **code** qui contienne le code secret, un champs **type_carte** qui est soit visa, soit mastercard.

Procédures, fonctions, passage par valeur et par référence

De facon simpliste, une procédure est un moyen de factoriser un code souvent utilisé et d'utiliser ce bloc simplement à un autre endroit du programme en l'appelant par son nom. En réalité c'est un mécanisme plus interessant que le simple recopiage car on peut utiliser le nom de la procédure dans le code même de la procédure que l'on est en train de définir.

```
procedure affiche_inf(entier n)
debut affiche_inf
 affiche(n)
 si (n>0) alors affiche_inf(n-1)
 finsi
fin affiche_inf
```

```
programme principal
debut principal
 affiche_inf(10)
fin principal
```

Une fonction se définit de la même facon sauf qu'elle retourne un résultat.

```
fonction somme(entier a,entier b) retourne entier
variable n un entier
debut somme
 n=a+b
 retourne n
fin somme

programme principal
variable x
debut programme
 x=somme(3,4)
fin programmme
```

Exercice 3 (TD-TP) :

Ecrire une fonction qui donne la valeur d'une carte à la belote.(Attention il y a un atout)

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

passage par reference et par valeur

Il reste une précision à donner sur les arguments passés aux fonctions et procédures

Exercice 4 :

Que fait le programme suivant ?

En pseudo

```
fonction f (valeur entier x)
            retourne entier
debut f
 x=x+1
 retourne x
fin f
fonction g (reference entier x)
            retourne entier
debut g
 x=x+1
 retourne x
fin g

programme principal
variable n entier
debut principal
 n=5;
 affiche (f(n))
 affiche (f(n))
 affiche (g(n))
 affiche (g(n))
fin principal
```

en C++

```
#include <iostream.h>

int f(int x){
  x=x+1;
  return x;
}

int g( int &x){
  x=x+1;
  return x;
}

void main(){
  int n;
  n=5;
  cout << f(n) << endl;
  cout << f(n) << endl;
  cout << g(n) << endl;
  cout << g(n) << endl;
}
```

Exercice 5 (TD) :

Ecrire une procedure qui puisse être utilisée à la place de la fonction de l'exercice 3. Avez vous une remarque d'ordre général ?

Exercice 6 (TD-TP) :

Supposons que le code d'une carte bleu est choisit au hasard lors de sa création, écrire une fonction **initialise_carte(... ma_carte, ... depot_initial, ... type_choisi)**.

Compléments: Pour genérer 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 aéatoires donne des résultats différents pour deux exécutions différentes, (ce qui serait pas mal), il faut être capable de l'initialiser differement 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 la librairie **stdlib.h**, et **time** est dans la librairie **time.h**.

Les tableaux, les constantes

En plus des types de bases, des enumerations et des structures, on peut aussi définir des blocs d'une certaine taille, composés de données de type identiques, ce sont les tableaux.

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

void main(){
 tableau t;
 int i;

 for(i=0;i<dim;i++) {
   t[i]=i;
 }
}
```

Remarquez que la numérotation va de 0 à 9, (ce qui fait 10 élements), et que les tableaux sont toujours passés par référence (on en verra l'explication dans quelques cours).

Exercice 7 :

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 placant dans b.

b) Ecrire une procédure $permute\_g(a)$ qui permute circulairement les éléments d'un tableau a, en les déplacant 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 facon, 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) On suppose que les tableaux a et b de taille n sont déjà triés. Ecrire une procédure $fusion(a, b, c)$ qui définit le tableau c de dimension $2n$ par insertion des des elements de a et de b.

En TP, on pourra aussi compléter les exercices sur la carte bleue, en définissant des morceaux de programmes pour le retrait, en fonction du nombre d'essais pour entrer le code, ou d'un decouvert borné maximum.

Correction

Exercice 1 :
```
#include <iostream.h>

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;
};

void main(){
  carte x;

  x.v =as;
  x.c=coeur;

}
```

Exercice 2 :
```
#include <iostream.h>

typedef enum geree_par {visa,mastercard};

typedef struct carte_bleue {
  int montant;
  int code;
  geree_par type_carte;
};

void main(){
  carte_bleue c;

  c.montant = 100;
  c.code=1418;
  c.type_carte=visa;

}
```

Exercice 3 :
```
#include <iostream.h>

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;
  }

}

void main(){
  carte x;

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

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

}
```

Exercice 4 : Il faut expliquer les passages par réference et par valeur. Si une majorité d'étudiants n'ont pas compris, ils peuvent mettre le nez dedans en TP et jouer à s'en convaincre. L'affichage obtenu est : 6 6 6 7

Exercice 5 : Attention ne pas oublier le break (comme moi).

```
#include <iostream.h>

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;
};


void valeur(carte c,couleur atout,int &resultat){
  switch (c.v) {
  case as : resultat= 11; break;
  case dix : resultat=10; break;
  case roi :  resultat=4;break;
  case dame : resultat=3;break;
  case valet : {
    if (c.c==atout)
       resultat=20;
    else  resultat=2;
    break;
  }
  case neuf : {
    if (c.c=atout) resultat= 14;
    else  resultat=0;
    break;
  }
  default : resultat=0;
  }

}

void main(){
  carte x;
  int resultat=1;

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


  valeur(x,pique,resultat);
```

```
  cout <<resultat << " " ;

  valeur(x,coeur,resultat);
  cout <<resultat << endl ;



}
```

Exercice 6 : Peut être qu'il n'est pas nécessaire de faire cet exercice completement. Seul le profil est interessant à préciser en TD, le reste peut être vu en TP.

```
#include <iostream.h>

#include <stdlib.h>
#include <time.h>

typedef enum geree_par {visa,mastercard};

typedef struct carte_bleue {
  int montant;
  int code;
  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=code_genere;

  return code_genere;
}

void main(){

  srand(time(NULL));

  carte_bleue c;

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

}
```

Exercice 7 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.h>

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;
  }
}

void main(){
  tableau a,b;

  init(a);
  init(b);

  a[0]=1; a[1]=2; a[2]=1;

  triangle(a,b);

  affiche(b);
}
```

Exercice 7 b) : Attention, la derniere case du tableau a pour numero $dim - 1$ (mon bug).

```
#include <iostream.h>

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;
}

void main(){
```

```
  tableau a;

  init(a);

  a[0]=1; a[1]=2; a[2]=1;

  permute_g(a);

  affiche(a);

}
```

Exercice 7 c) : Là il ne faut rien ecraser, la boucle se deroule donc dans l'autre sens.

```
#include <iostream.h>

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;
}

void main(){
  tableau a;

  init(a);

  a[0]=1; a[1]=2; a[2]=1;

  permute_d(a);

  affiche(a);

}
```

Exercice 7 d) : La seule remarque c'est que ca se fait directement, sans rien ecraser si on lit de droite a gauche.

```
#include <iostream.h>

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;
}

void 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 7 e) :

```
#include <iostream.h>

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++;
    };
  }

}

void 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);
}
```