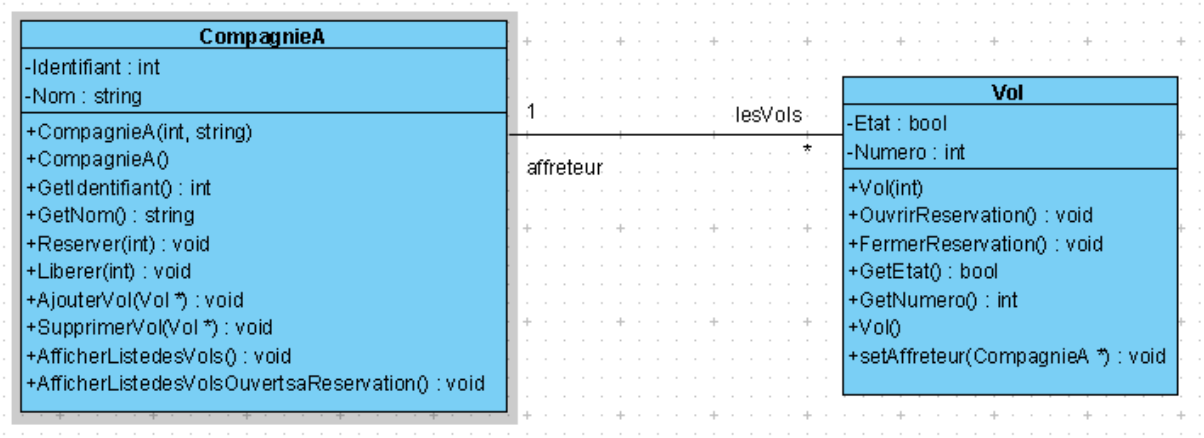
Associations UML et C++

Exercice 3

Soit le diagramme de classe suivant :



Rappel :

Les associations entre classes sont généralement représentées par des pointeurs (parfois des références). Les classes associées possèdent en attribut un ou plusieurs pointeurs (ou références) vers l'autre classe.

Le nombre de pointeurs dépend de la cardinalité. Lorsque la cardinalité est 1 ou 0..1, il n'y a qu'un seul pointeur de l'autre classe

Dans le cadre d'une multiplicité 0..n l'association ne peut plus être un tableau car sa taille est fixée à la création de ce dernier. Le tableau n'est pas dynamique.

Pour contourner ce problème, il convient d'utiliser un **conteneur** (une **collection**).

Les collections sont des objets qui permettent de gérer des ensembles d'objets.
Ces ensembles de données peuvent être définis avec plusieurs caractéristiques : la possibilité de gérer des doublons, de gérer un ordre de tri, etc. ...

Chaque objet contenu dans une collection est appelé un élément.

Il existe plusieurs types de collection en C++ : vector, list, map ….Il appartiennent à la bibliothèque STL.

Idée générale de la bibliothèque STL:

On ne se préoccupe pas de la nature de ce que l'on manipule : des entiers, des réels, des chaînes de caractères ou des types « maisons » comme Piece, Personne....

- Les conteneurs règlent l'organisation des objets.
- Les itérateurs sont chargés de l'accès aux objets.
- Le traitement est effectué par les fonctions génériques de <algorithm>.

LES CONTENEURS

Il existe 2 familles de conteneurs:

- de séquence:
  vector, deque et list
- Associatifs:
  set, multiset, map et les multimap

Le vector est le conteneur de remplacement du tableau. Il peut grossir pendant l'exécution.

Il y a 2 types d'implémentation en mémoire:

- Une zone mémoire continue est allouée (comme un tableau) : vector,deque,
- La liste: les objets sont dispersés dans la mémoire et relié entre eux par un couple de pointeurs 'Précédent' 'Suivant'. Pour un nombre d'objets identiques une liste prend souvent plus de place en mémoire.

Tous les conteneurs disposent de 3 fonctions:

- begin() renvoie l'itérateur sur le 1 objet
- end() renvoie l'itérateur sur l'itérateur juste après le dernier objet
- empty() renvoie le booléen true si le conteneur est vide

LES ITERATEURS <ITERATOR>

Un itérateur est un objet permettant l'accès aux objets d'un conteneur. Il peut être vu comme une variable de position. Il supporte l'opération d'incrémentation qui permet de «passer» à l'objet référencé suivant. Il y a différents types d'itérateurs.

L'itérateurs de type « bidirectional_iterator » s'applique aux conteneurs: list, set, multiset, map, multimap.

L'itérateur de type « random_acces_iterator » est le plus puissant. Ils'applique aux conteneurs de type: vector, deque.

Les itérateurs adaptateurs permettent de :

- Lier un flux de données à un conteneur,
- Réaliser les opérations d'insertion d'un objet dans un conteneur.

LES FONCTIONS <ALGORITHM>

C'est un jeu de 70 fonctions traitant les algorithmes les plus connus: la création, la copie, la suppression, le remplacement, la recherche avec un critère, le tri. Elles peuvent:

- S'appliquent à tous les objets d'un conteneur ou seulement une partie.
- Opérer sur 2 conteneurs de nature différents à condition qu'ils contiennent des objets de même type.

Exemple :

```
#include <algorithm>    // std::find
#include <vector>       // std::vector

        std::vector<int> myvector (myints,myints+4);
        std::vector<int>::iterator it;

        // iterator to vector element:
        it = find (myvector.begin(), myvector.end(), 30);
```

LE CONTENEUR VECTOR

Le conteneur stocke des objets de même type dans un vecteur. Il permet un accès direct aux objets comme pour un tableau mais par contre :

- Il peut grossir de manière dynamique, pendant l'exécution, par une réallocation mémoire du vector.
- Il met à disposition des services comme l'affection de vecteur à vecteur *vec1=vce2*, la comparaison de vecteurs *vec1 == vec2 vce1<vec2*, des traitements (insérer un objet à la fin *push_back()* , vider un conteneur *clear()* , tester si un vector est vide *empty()…*

1. Déclarer un vector : vector<type> identifiant

```
#include <vector> // Entête pour vector
using namespace std;

int main(){

vector<float> vecFloat; //Définit un vector de flottants vide
vector<int> vecInt(10); //Définit un vector de 10 entiers
vector<char> vecChar(tab,tab+4);//Définit un vector de 4 caractères initialisé

-------------- ; //avec la séquence d'un tableau
}
```

Types contenus dans un vector : Le types de base du langage C++ ou propriétaire les structures, les classes

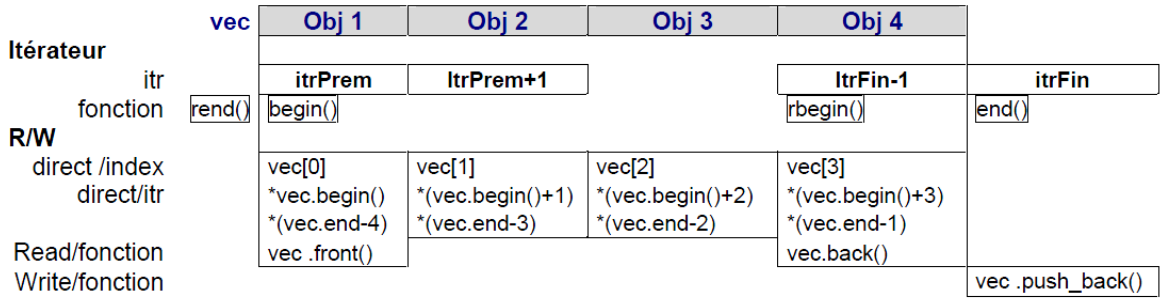
2. Méthodes et opérateurs s'appliquant au vector

assign() Affecte, depuis le début, une valeur à plusieurs objets
vec.assign(2, 77); // les 2 premiers reçoivent 77
Affecte, depuis le début, la séquence d'un autre conteneur
vec.assign(tab, tab+5); // vec[0 à 5] <- tab[0 à 5]
at(index)=val Idem à [index] mais plus sur : Vérifie la type de val et les limites (prévoir Trait Error)
vec.at(1)=1000; // Affecte 100 au 2ème
back() Retourne la valeur du dernier objet.
begin() Retourne l'itérateur Premier : position du 1er objet.
capacity() Retourne la capacité, le nombre d'objets possibles de stocker dans le vector
clear() Efface tous les objets du vector
empty() Teste si le vector est vide (true si pas d'objet size()==0)
erase(itrPrem, itrDer) Efface les objets de la séquence définie par les itérateurs
vec.erase(vec.begin+1(), vec.end()-6);
Efface l'objet pointé par l'itérateur
vec.erase(vec.begin+4());
end() Retourne l'itérateur Fin : position suivante après le dernier objet du vector

front() Retourne la valeur du 1er objet
insert() Insérer un objet à une position
vec.insert(vec.begin()+2, 33); // *(Début+2) =33
Insérer une série d'objets de même valeurs, à partir d'une position
vec.insert(vec.begin()+5, 3, 1789);
// A partir de Début+5, insère 3 objets de valeur 1789
Insérer, à partir la position, une séquence d'un autre conteneur.
vec.insert(vec.begin()+7, vec3.begin()+2, vec3.end()-6);

pop_back() Retire le dernier objet et retourne sa valeur
push_back(val) Ajoute un objet à la fin et lui affecte la valeur « val »
reserve(nbre) Réserve de la mémoire pour stocker « nbre » objets. Cette fonction permet de seulement augmenter la capacité avec éventuellement une réallocation mémoire du vector.
resize (nbre) La capacité est adaptée pour stocker « nbre » d'objets. La capacité du vector peut être diminuée avec cette méthode.
size() Retourne le nombre d'objets contenus dans le vector.
swap Echange des valeurs avec un autre conteneur (même type d'objets)
vec.swap(vec3) ;
[ ] Accès direct indexé à la valeur d'un élément du vector (comme un tableau)
***** Opérateur d'indirection : *vec1.begin() retourne la valeur pointée par begin()
= Affectation entre vecteurs : vec1=vce2 ;
==, <,>,.etc.. Tous les opérateurs de comparaison : if(vec1 !=vec2) ---- ;

3. Accès aux objets et itérateurs



Séquence et Itérateurs

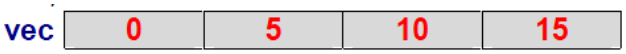
La séquence des objets contenus dans un vector est définie par 2 itérateurs : itrDébut et itrFin.
Les méthodes begin() et end() retourne la valeur de ces itérateurs début et Fin.
Le dernier objet se trouve à end()-1
Les fonctions rbegin() et rend() renvoie deux itérateurs inverses qui permettent de traverser la séquence à l'envers de la Fin vers le Début.

Accéder et Parcourir

Accès direct aux valeurs : `vec[1]=20000 ;`
Accès direct par les itérateurs : `*vec.begin()=10 ; *(vec.begin()+2)=33 ;`

Parcourir par les index et affecter des valeurs :

```
for (int i=0;i<vec.size();i++) vec[i]=i*5;
```



Parcourir par les itérateurs et affecter des valeurs:

```
vector<int>::iterator itr;
for (itr=vec.begin();itr!=vec.end();itr++) *itr= *itr*2;
```

Parcourir par les itérateurs en sens inverse :

```
vector<int>::iterator_revers itrRvr; // Itérateur rebours
for (itrRvr=vec.rbegin();itrRvr!=vec.rend();itrRvr++) *itrRvr=*itrRvr-10;
```

4. Ajouter des objets

Le conteneur vector insère naturellement les nouveaux objets par la fin avec la fonction push_back(). Si le vector à une capacité trop petite, il est automatiquement redimensionné. La capacité peut doubler par l'ajout d'un objet, voir § «Taille et Capacité» Attention, l'implémentation mémoire du vector peut changer.

```
vec.push_back(555);
```

vec	-10	0	10	20	555

5. Taille et capacité

La capacité : C'est le nombre d'objets que peut contenir le vector. Représente l'occupation mémoire. Occupation mémoire (octets)= Capacité x Taille du type d'objets (octets). La méthode *capacity()* retour sa valeur.

La taille : C'est le nombre d'objets réels contenus dans la séquence irtDébut/begin() irtFin/end()-1, d'un vector.

La méthode size() retour sa valeur.

La fonction push_back() redimensionne automatiquement le vector, si besoin est

```
vector<int> vec;
  for(int i=0;i<10;i++){
      vec.push_back(i+2); // Insère à la fin un obj de valeur i+2
      cout<<"C "<<vec.capacity() // Affiche la Capacité : 16
      <<" T "<<vec.size() // Affiche la Taille : 10
      <<" Val "<<*(vec.end()-1) // Affiche la Valeur du Dernier
      << endl;
  }
```

vec	2	3	4	5	6	7	8	9	10	11						
Taille	1	2	3	4	5	6	7	8	9	10						
Capacité	1	2	4	4	8	8	8	8	16	16						16

L'itérateur Fin/end() est positionné après le dernier objet et non à la fin du conteneur. Il permet d'exploiter la partie utile du conteneur vector. Comment fixer la capacité du vector ?

- A la déclaration

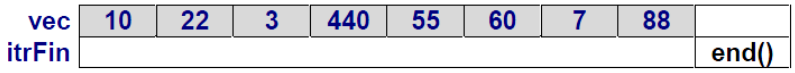
```
vector<int> vec // Size 0, Capacité 0
vector<int> vec1(10) // Size 10, Capacité 16
vector<int> vec2(debut, debut+5) // Size 5, Capacité 8
```

- Pendant l'exécution

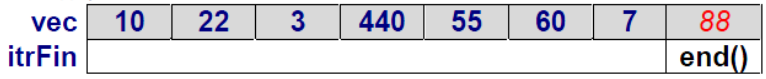
```
vec2.push_back(Valeur) // Automatique voir ci-dessus
vec.insert(vec.begin(), 2) // Automatique comme push_back()
vce1.reserve(17) // Nouvelle Size 17, Capacité 32
```

6. Supprimer des objets

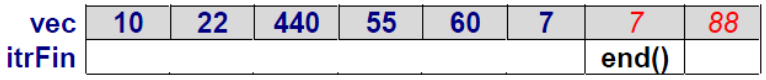
La suppression consiste à déplacer l'itérateur Fin/end(). Attention, elle ne modifie pas la capacité du vector mais uniquement sa taille.



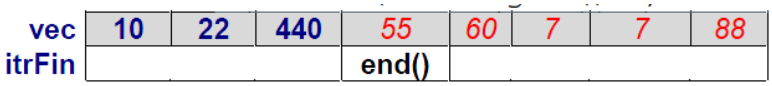
Supprimer le dernier : `vec.pop_back();`



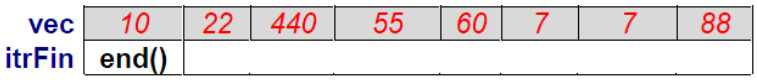
Supprimer un objet : `vec.erase(vec.begin()+2) ;`



Supprimer une séquence d'objets entre itr1 et itr2 :
vec.erase(vec. begin()+3,vec.end());



Supprime tous les objets : `vec.clear();`



7. EXEMPLES

Accès à la valeur avec l'opérateur [ ], un vector comme un tableau

```
#include <iostream>
#include <vector>
using namespace std;
// data source
int tab[10]={100,22,33,4,555,600,7,80,9,11};
int main(void){
      // Définit un vector initialisé avec les valeurs de tab à (tab+10)-1
      vector<int> vec1(tab, tab+10);
      // Accéder aux membres avec l'opérateur []
      cout<<"vec1 Premier:"<<vec1[0] <<" Suivant:"<<vec1[1] << "Dernier:"
      << vec1[9]<<endl;
      vec1[1]=20000; // Modifie un objet
      cout<<"vector vec1 : ";
      // Parcourir pour faire quelque chose connaissant le nombre d'objets
      for (int i=0; i<vec.size(); i++) cout<<vec1[i]<<", ";
      cout<< endl;
      return 0;
}
```

Résultat

```
vec1 Premier:100 Suivant:22 Dernier:11
vector vec1 : 100, 20000, 33, 4, 555, 600, 7, 80, 9, 11,
```

L'accès à un objet du vector est possible avec l'opérateur [ ] comme pour un tableau et peut être parcouru de la même manière.

Opérations entre vectors : Affecter, Comparer

```
#include <iostream>
#include <vector>
using namespace std;
// data source
int tab[10]={100,22,33,4,555,600,7,80,9,11};
int main(void){
      // Définit un vector initialise avec les valeurs de tab à (tab+10)-1
      vector<int> vec1(tab, tab+10);
      // Définit un vector de 10 entiers
      vector<int> vec2(10);
      // Affectation vector à vector vec2<-vec1
      vec2=vec1; // Affecte vec1 à vec2
      vec2[0]=10; // Modifie vec2 en affectant 10 au 1er
      cout<<"vector v2 : ";
      // Comparer 2 vectors
      if(vec2==vec1) // Teste l'égalité
      cout<<"vec2=vec1"<<endl;
      else if (vec2<vec1) // Teste si <
      cout<<"vec2<vec1"<<endl;
      else
      cout<<"vec2>vec1"<<endl;
      cout<<endl;
      return 0;
}
```

Faire «grossir» un vector en ajoutant des objets. Accès à la valeur par l'itérateur

```
#include <iostream>
#include <vector>
using namespace std;
int main(void){
      // Définit un vector d'entier vide
      vector<int> vec3;
      // Définit un itérateur sur un vector d'entier
      vector<int>::iterator itr;
      int taille =0;
      int nbreObjs=0;
      taille=vec3.capacity(); // Donne la capacité du vector
      nbreObjs=vec3.size(); // Donne le nombre d'objs dans le vector
      cout<<"Capacite "<<taille<<" Nbre d'objs "<<nbreObjs<<endl;
      // Faire grossir un vector en ajoutant un obj avec push_back(val)
      // Ajouter des objets à la fin du vector - Le vector s'étire
      vec3.push_back(10); // Ajoute 1 Obj a la Fin
      vec3.push_back(2); // Ajoute 1 Obj a la Fin
      vec3.push_back(30); // Ajoute 1 Obj a la Fin
      cout<<'\t'<<"vector vec3 : ";
      // Parcourir pour faire quelque chose en utilisant un itr
      // Pour (itr de itrDebut_vec3 à itrDebut_vec3-1, au pas de 1,
      // afficher valeur référencée par itr (* Renvoie la valeur
      référencée/itr)
      for (itr=vec3.begin(); itr<vec3.end(); itr++) cout<<*itr<<", ";
      taille=vec3.capacity();
      nbreObjs=vec3.size();
      cout<<endl<<" Capacite "<<taille <<" Nbre d'objs "<<nbreObjs<<endl;
      cout<< endl;
      return 0;
}
```

Résultat

```
Capacite 0 Nbre d'objs 0
vector vec3 : 10, 2, 30,
Capacite 4 Nbre d'objs 3
```

il est possible d'accéder à la valeur d'un objet en déréférençant l'itérateur avec l'opérateur *. Un vector peut grossir pendant l'exécution en ajoutant des objets à la fin avec la fonction push_back (). Le vector se redimensionne automatiquement. Remplace le tableau à création dynamique.
Si la capacité du vector est trop petite lors de l'ajout d'objet avec les méthodes push_back() ou insert(), une zone mémoire est réallouée au vector pour augmenter sa capacité Voir exemple § « Taille et capacité ». Alors, l'implémentation mémoire du vector est changée, donc les itérateurs associés vector ont modifiées. Ceci peut être pénalisant, si utilise ailleurs dans le programme les itérateurs ans du vector.

Faire «maigrir » un vector en supprimant des objets

```
#include <iostream>
#include <vector>
using namespace std;
// data source
int tab[10]={100,22,33,4,555,600,7,80,9,11};
int main(void){
      // Définit un vector initialise avec les valeurs de tab à (tab+10)-1
      vector<int> vec1(tab, tab+10);
      // Définit un vector de 10 entiers
      vector<int> vec2(10);
      // Définit un itérateur sur un vector d'entier
      vector<int>::iterator itr;
      int taille =0;
      int nbreObjs=0;
      vec2=vec1; // Affecte vec1 à vec2
      taille=vec2.capacity(); // Donne la capacité du vector
      nbreObjs=vec2.size(); // Donne le nombre d'objs dans le vector
      cout<<" Capacite "<<taille<<" Nbre d'objs "<<nbreObjs<<endl;
      // Faire maigrir un vector en supprimant un objs avec pop_back()
      // Efface des objs à la fin du vector - Le vector rétrécit
      vec2.pop_back(); // Supprime 1 Obj a la fin
      vec2.pop_back(); // Supprime 1 Obj a la fin
      vec2.erase(vec2.end()-4,vec2.end());// Supprime 4 Objs a la fin
      cout<<'\t'<<"vector vec2 : ";
      // Parcourir pour faire quelque chose en utilisant un itr
      // Pour (itr de itrDebut_vec2 à itrFin_vec2-1, au pas de 1,
      // afficher la valeur référencée par itr
      for (itr=vec2.begin(); itr<vec2.end(); itr++) cout<<*itr<<", ";
      cout<< endl;
      taille=vec2.capacity();
      nbreObjs=vec2.size();
      cout<<" "<<"vec2 Capacite "<<taille<<" Nombre d'objs "<<nbreObjs<<'
      ';
      // Attention la capacité cad l'occupation en mémoire ne change pas
      // Ce qui change c'est irtFin
      cout<<endl<<'\t'<<"vector vec2 : ";
      for (itr=vec2.begin(); itr<vec2.end()+6; itr++) cout<<*itr<<", ";
      cout<< endl;
      vec2.clear();
      cout<<"vec2.clear() "<<endl;
      taille=vec2.capacity();
      nbreObjs=vec2.size();
      cout<<" "<<"vec2 Capacite "<<taille<<" Nombre d'objs
      "<<nbreObjs<<endl;
      for (itr=vec2.begin(); itr<vec2.end(); itr++) cout<<*itr<<", ";
```

```
	cout<< endl;
	return 0;
}
```

Résultat

```
Capacite 10 Nbre d'objs 10
vector vec2 : 100, 22, 33, 4,
vec2 Capacite 10 Nombre d'objs 4
vector vec2 : 100, 22, 33, 4, 555, 600, 7, 80, 9, 11,
vec2.clear()
vec2 Capacite 10 Nombre d'objs 0
```

Supprimer des objets consiste à déplacer l'itérateur Fin afin de disposer de la séquence utile du vector.Ce qui diminue (maigrit) c'est de taille mais pas la capacité.

1. Ecrire le nouveau fichier de déclaration de la classe CompagnieA.

2. Réécrire les méthodes suivantes :

- **CompagnieA()**

- **~CompagnieA()**

- **Public void AjouterVol(Vol *vol)**

Cette méthode permet l'ajout d'un objet Vol dans le vector lesVols. Cette méthode provoque l'affichage du message suivant : ajout du vol vol1.. dans le vector lesVols.

- **Public void SupprimerVol(Vol *vol)**

Cette méthode permet la suppression d'un objet Vol du vector lesVols. Cette méthode provoque l'affichage du message suivant : suppression du vol vol1.. du vector lesVols .

- **Public void AfficherListedesVols ()**

Cette méthode liste les objets Vol présent dans le vector lesVols. Cette méthode provoque l'affichage du message suivant :

Liste des vols enregistres :

Vol 1, Vol 2, Vol3 , ….

- **Public void AfficherListedesVolsOuvertsaReservation ()**

Cette méthode liste les objets Vol présent dans le vector lesVols dont la réservation est ouverte. Cette méthode provoque l'affichage du message suivant :

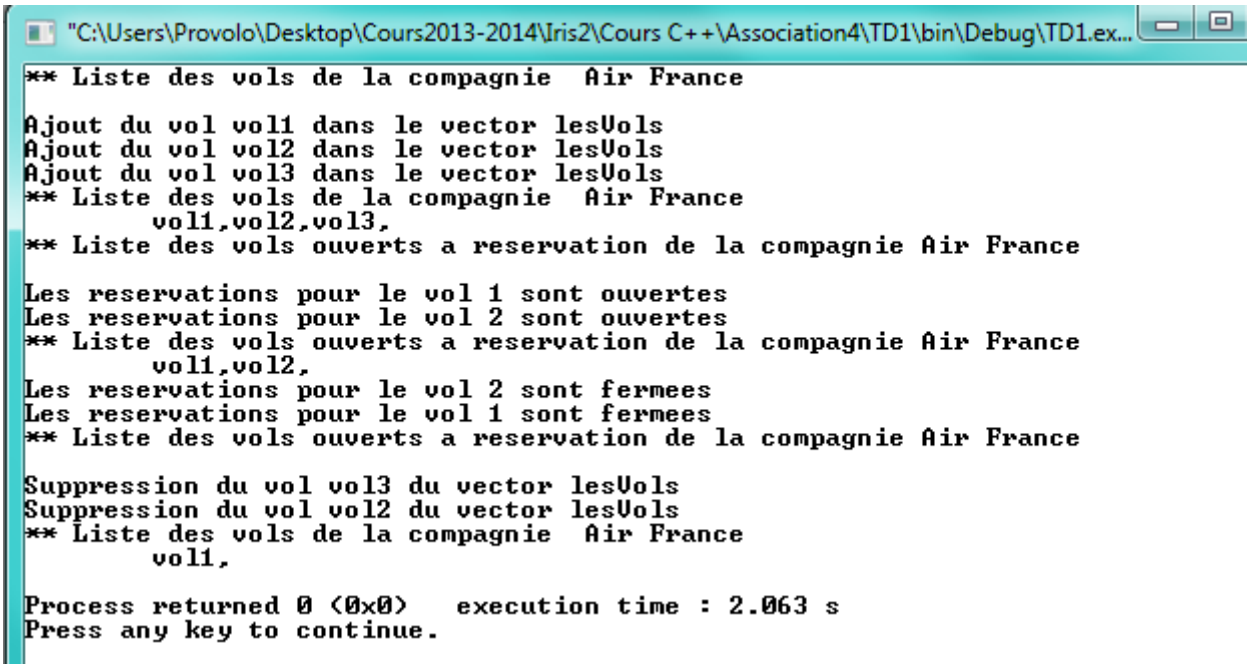
Liste des vols ouverts à réservation

Vol 1,….

- Ré-ecrire les méthodes **void Liberer (int numero_vol)** et **Reserver(int numero_vol)** afin de prendre en compte le vector lesVols

Exemple de fichier main.cpp

```
#include <iostream>
#include "vol.h"
#include "compagniea.h"

using namespace std;

int main()
{
    Vol *vol1, *vol2, *vol3;
    CompagnieA  * CA1;

    vol1 = new Vol(1);
    vol2 = new Vol(2);
    vol3 = new Vol(3);
    CA1 = new CompagnieA(1,"Air France");

    CA1->AfficherListedesVols();

    CA1->AjouterVol(vol1);
    CA1->AjouterVol(vol2);
    CA1->AjouterVol(vol3);

    CA1->AfficherListedesVols();

    CA1->AfficherListedesVolsOuvertsaReservation();

    CA1->Reserver(1);
    CA1->Reserver(2);
    CA1->AfficherListedesVolsOuvertsaReservation();
    CA1->Liberer(2);
    CA1->Liberer(1);

    CA1->AfficherListedesVolsOuvertsaReservation();

    CA1->SupprimerVol(vol3);
    CA1->SupprimerVol(vol2);
    CA1->AfficherListedesVols();



    return 0;
}
```


"C:\Users\Provolo\Desktop\Cours2013-2014\Iris2\Cours C++\Association4\TD1\bin\Debug\TD1.ex...
** Liste des vols de la compagnie Air France
Ajout du vol vol1 dans le vector lesVols
Ajout du vol vol2 dans le vector lesVols
Ajout du vol vol3 dans le vector lesVols
** Liste des vols de la compagnie Air France
vol1,vol2,vol3,
** Liste des vols ouverts a reservation de la compagnie Air France
Les reservations pour le vol 1 sont ouvertes
Les reservations pour le vol 2 sont ouvertes
** Liste des vols ouverts a reservation de la compagnie Air France
vol1,vol2,
Les reservations pour le vol 2 sont fermees
Les reservations pour le vol 1 sont fermees
** Liste des vols ouverts a reservation de la compagnie Air France
Suppression du vol vol3 du vector lesVols
Suppression du vol vol2 du vector lesVols
** Liste des vols de la compagnie Air France
vol1,
Process returned 0 (0x0) execution time : 2.063 s
Press any key to continue.