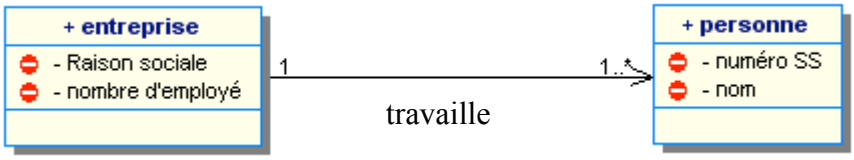
POO STL
Cours 4

Ce cours permet de voir l'utilisation des conteneurs, des itérateurs, et des algorithmes. Ces nouveaux éléments C++ sont disponibles dans la Standard Template Library (STL). La STL permet au développeur C++ de disposer de composants prédéfinis. Tous ces composants sont membres de l'espace de nom std.

Pour cela partons du problème suivant :

Une personne travaille dans une seule entreprise. Une personne possède un numéro de SS et un nom.
L'entreprise emploie plusieurs personnes. Elle possède une raison sociale et un nombre d'employé.

a. Réaliser le DCL



Nous savons comment traiter le problème poser par le DCL pour définir les classes en langage de programmation.

b. Ecrire l'interface des classes correspondantes au DCL

Supposons que chaque classe dispose :

- D'un constructeur paramétré
- D'un destructeur
- D'une méthode affiche qui affiche les valeurs de données membres

c. Ajouter ces méthodes à vos classes et écrire l'implémentation.

1. Utilisation des conteneurs

Nous avons introduit avec nos transformations de DCL vers les classes que certaines associations pouvaient donner naissance à des tableaux à une ou plusieurs dimensions. Nous pouvons associer ces tableaux à des collections d'objets.

Les collections d'objets peuvent être traitées à l'aide des conteneurs.
Un conteneur est un objet qui contient d'autres objets.

Les conteneurs disposent de fonctions :

- Insert : insertion d'élément dans le conteneur
- Erase : effacement d'un élément
- Clean : effacement de tous les éléments
- Begin : renvoi un itérateur positionné sur le premier élément du conteneur
- End : renvoi un itérateur positionné sur la marque de dépassement du conteneur (juste après le dernier élément).

Les conteneurs de base contiennent des objets de même type, mémorisés linéairement.
La catégorie des conteneurs propose trois séquences de base :

- Vector (vecteur), bon compromis.
- List (liste), optimisée pour les insertions et suppressions en milieu de séquence
- Deque (The name *deque* is pronounced "deck", and stands for "double-ended queue."). Optimisée pour les opérations en début et fin de séquence.

Des itérateurs sont associés à ces séquences. Ils permettent de parcourir les éléments des conteneurs.

La séquence vector contient les fonctions supplémentaires :

- Push_back : insertion d'un élément en queue de vecteur
- Pop_back : suppression d'un élément en queue de vecteur
- Push_front : insertion d'un élément en début de vecteur
- Size : taille du vecteur ou nombre d'élément
- [] : accès direct aux élément du vecteur.

Le conteneur vector est utilisable avec la class vector (#include <vector>).

Syntaxe :

```
Vector<type> nomVector ;
Vector<objets*> nomVector ;

Vector<int> v(8); // conteneur de 8 entiers
Vector<personne*> travaille; // conteneurs de pointeurs sur personnes
```

Le conteneur vector est l'équivalent des Vector et ArrayList Java.

d. Remplacer le tableau à plusieurs dimensions dans votre classe entreprise par une collection nommée travaille qui stocke des objets personne à l'aide d'une séquence de base vector.

Vous placerez cette collection en private dans votre classe entreprise est vous ajouterez les méthodes suivantes.

- ajoutPersonnel(personne &p) : ajoute la personne p à la liste des employés.
- listePersonnel() : permet de connaître le nombre d'employé et les informations sur chaque personne de l'entreprise.

2. Les itérateurs

Les conteneurs peuvent créer des itérateurs. En particulier les conteneurs vector, list et deque.

Syntaxe :

Vector<type> ::iterator it ;
Vector<objets*> ::iterator it ;

Vector<int>::iterator it;
Vector<personne*>::iterator it;

e. associer un itérateur à votre conteneur travaille dans votre classe entreprise. Vous placerez cet itérateur en membre privé.

f. réécrire le code des méthodes ajoutPersonnel et listePersonnel à l'aide de l'itérateur que vous venez d'implémenter.

3. Les algorithmes

Cette catégorie d'objet contient des composants réalisant des traitements sur les conteneurs. Ces composants sont déclarés dans la classe algorithm (#include <algorithm>). Il existe deux sortes d'algorithmes : ceux qui modifient le conteneur traité et ceux qui ne le modifient pas.

Algorithmes ne modifiant pas le conteneur :

- Find : recherche
- For_each : appliqué une fonction à chaque élément
- Count : compte les éléments équivalent à celui passé en paramètre
- Count_if : compte les éléments avec une condition

Algorithmes modifiant le conteneur :

- Replace : remplace les anciens éléments par de nouveau éléments
- Replace_if : remplace avec condition

Voyons l'algorithme de recherche find qui va nous permettre de trouver un élément dans notre conteneur.

Syntaxe :

Iterator find(iterator first, iterator last, const T &value) ;
It = find(travaille.begin(),travaille.end(),&p);

L'algorithme find renvoie un itérateur sur le premier élement à égal à la valeur passée en paramètre (value)
Les éléments parcourus sont ceux du conteneur associé aux itérateurs fisrt et last, dans l'intervalle [first,last].

g. Ecrire deux méthodes supplémentaires dans la classe entreprise :

- licenciementPersonne(&p) : qui permet de supprimer de la liste des employés de l'entreprise concernée, la personne p.
- licenciementGlobal() : qui permet de supprimer toutes les personnes de la liste de l'entreprise.

Dans ces deux méthodes vous utiliserez les itérateurs, les algorithmes, et les fonctions disponibles sur les séquences de base vector.

h. Ecrire l'application theApp qui permet d'utiliser les méthodes des classes que vous venez de définir :

créer une personne A numéro SS : « 189323232 » nommée « fred »
créer une personne B numéro SS : « 235689897 » nommée « bob »
créer une personne C par défaut
créer une personne D numéro SS : « rerereerer » nommée « deb »
créer l'entreprise Z « SA latour » zéro employé
Afficher les informations sur l'entreprise Z.
Ajouter les personnes A, B, C, D dans le personnel de l'entreprise Z
Lister le personnel de l'entreprise Z.
Licencier la personne B de l'entreprise Z
Lister le personnel de l'entreprise Z
Effectuer un licenciement global
Lister le personnel de l'entreprise Z

i. Construire un tableau à deux dimensions de personnes est créer les méthodes suivantes :

La première dimension permettra de stocker 100 personnes.

- gestionPersonnel(personne &p) : ajout d'une personne sur une dimension du nouveau tableau
- listeGestionPersonnel() : affichage du contenu du tableau à une deux dimension.

j. Ajouter l'appel de ces deux dernières méthodes dans votre application

Ajouter les personnes A et B dans le tableau à deux dimensions.
Afficher les personnes A et B soit le contenu de votre tableau à deux dimensions.

```
#ifndef _ENTREPRISE_H
#define _ENTREPRISE_H

#include <iostream>
#include <vector> // pour utiliser le conteneur vector et l'itérateur
#include <algorithm> // pour utiliser find


using namespace std; // espace de nommage standard

#include "personne.h" // cela est plus puissant permet d'utiliser les
méthodes de personne dans entreprise
//class personne; pas suffisant ne permet d'utiliser affiche de personne
dans listeEmploye de entreprise

class entreprise
{

      private :

            char *chRaisonSociale;
            int intNombreEmploye;
            // on peut mettre le vecteur en public ou en privé cela ne
change rien à part pour l'encapsulation
            //vector<personne*> travaille; // au lieu d'utiliser un tableau
à deux dimensions on utilise un vecteur de pointeur sur personne
            vector<personne*>::iterator it; // définition d'un itérateur
            vector<personne*> travaille; // au lieu d'utiliser un tableau à
deux dimensions on utilise un vecteur de pointeur sur personne

            vector<vector<personne*> >travaille2D ; // tableau à 2
dimensions de personnes

      public :


            entreprise(char *chRaisonSociale = "", int intNombreEmploye =
0);
            void affiche();
            ~entreprise();
            // on peut mettre le vecteur en public ou en privé.


            //personne **travaille;
            void listePersonnel();  // affichage du contenu du tableau à une
                                    dimension
            void ajoutPersonnel(personne &p);   // ajout d'une personne dans
                                                   le tableau 1 dimension
            void gestionPersonnel(personne &p); // remplissage du tableau à
                                                   2 dimensions
            void listeGestionPersonnel(); // affichage tableau 2 dimensions
            void licenciementPersonne(personne &p);   // suppression d'une
                                                         personne dans le
                                                         tableau 1 dimension
            void licenciementGlobal();    // suppression de toutes les
                                          personnes dans le tableau 1
                                          dimension

};

#endif /* _ENTREPRISE_H */
```

```
#ifndef _PERSONNE_H
#define _PERSONNE_H

#include <iostream> // nécessaire pour les méthodes de personne
using namespace std;

//class entreprise; permet d'utiliser le type entreprise pour le tableau

class personne
{

      private :

            char *chNumeroSs;
            char *chNom;


      public :


            personne(char *chNumeroSs = "", char *chNom = "");
            void affiche();
            ~personne();
            //entreprise *travaille;

};

#endif /* _PERSONNE_H */
```

```
#ifndef _ENTREPRISE_CPP
#define _ENTREPRISE_CPP

#include "entreprise.h"

class personne;

entreprise::entreprise(char *chRaisonSociale, int intNombreEmploye)
{
	//this->chRaisonSociale = new char[50];
	this->chRaisonSociale = chRaisonSociale;
	this->intNombreEmploye = intNombreEmploye;
	travaille2D.resize(100);
}

void entreprise::affiche()
{
	cout<<"Raison sociale : "<<this->chRaisonSociale<<" Nombre employes :
"<<this->intNombreEmploye<<endl;
}

void entreprise::ajoutPersonnel(personne &p)
{
	this->intNombreEmploye++;
	travaille.push_back(&p); //permet une insertion d'un élément en queue
de vecteur pas besoin d'itérateur
	// ce qui donne à l'affichage l'ordre d'insertion
	//                A
	//          A,B
	//      A,B,C
	//    A,B,C,D
	//it=travaille.begin(); // nécessité d'un itérateur avec insert.
	//travaille.insert(it,&p); //permet une insertion d'un élément en
début de vecteur;
	// ce qui donne à l'affichage l'ordre inverse d'insertion
	// A
	// B,A
	// C,B,A
	// D,C,B,A
}

void entreprise::gestionPersonnel(personne &p)
{
	travaille2D[intNombreEmploye].push_back(&p); // ajout d'une personne
par dimension
	this->intNombreEmploye++;
	//travaille2D[0].push_back(&p); // ajout de toutes les personnes sur
la même dimension 0
}

void entreprise::listeGestionPersonnel()
{
	cout<<"******** Pesonnel ************************"<<endl;
	personne *p;
	int dim1 =(unsigned int)travaille2D.size();
	int dim2;
	for (int i = 0 ; i < dim1 ; i++)
	{
```

```
            dim2 = (unsigned int) travaille2D[i].size();

            for (int y = 0 ; y  < dim2 ; y++)
            {
                  p=(personne *) travaille2D[i][y];
                  p->affiche();

            }
      }
}
void entreprise::listePersonnel()
{
      cout<<"Nombre d'employes :"<<this->intNombreEmploye<<endl;
      // solution sans itérateur qui marche
      //int nombre=(unsigned int)travaille.size();
      //personne *p;
      //for (int i =0; i<nombre;i++)
      //{
      //    //cout<<travaille[i]; renvoir l'adresse de l'objet
      //    p=(personne *)travaille[i];
      //    cout<<p;
      //    p->affiche();
      //}
      //il est possible d'écrire le code suivant équivalent */

      // solution avec itérateur
      personne *p;
      for (it=travaille.begin(); it!=travaille.end(); it++)
      {
            //cout << *it << endl; affiche les adresse des objets
            p=(personne *)*it;
            p->affiche();
      }



}

void entreprise::licenciementPersonne(personne &p)
{
      this->intNombreEmploye--;
      it = find(travaille.begin(),travaille.end(),&p);
      travaille.erase(it);
}

void entreprise::licenciementGlobal()
{
      this->intNombreEmploye=0;
      travaille.clear();
}

entreprise::~entreprise()
{

}


#endif /* _ENTREPRISE_CPP */
```

```
#ifndef _PERSONNE_CPP
#define _PERSONNE_CPP


#include "personne.h"

personne::personne(char *chNumeroSs, char *chNom)
{
	//this->chNumeroSs = new char[17];
	this->chNumeroSs = chNumeroSs;
	//this->chNom = new char[50];
	this->chNom = chNom;
}

void personne::affiche()
{
	cout<<"n-SS : "<<this->chNumeroSs<<" Nom : "<<this->chNom<<endl;
}

personne::~personne()
{}

#endif /* _PERSONNE_CPP */

TheAPP.cpp
#include "stdafx.h"
#include "personne.h"
#include "entreprise.h"



int _tmain(int argc, _TCHAR* argv[])
{
	personne A("189323232","seb");
	personne B("235689897","bob");
	personne C;
	personne D("rerereeerer","deb");
	entreprise Z("SA latour",0);
	Z.affiche();
	Z.ajoutPersonnel(A);
	Z.ajoutPersonnel(B);
	Z.ajoutPersonnel(C);
	Z.ajoutPersonnel(D);
	Z.listePersonnel();
	Z.licenciementPersonne(B);
	Z.listePersonnel();
	Z.licenciementGlobal();
	Z.listePersonnel();
	//travaille.push_back(A); // impossible car iconu

	Z.gestionPersonnel(A);
	Z.gestionPersonnel(B);
	Z.listeGestionPersonnel();
	return 0;
}
```