TD-TP *LES CONTENEURS « list pair map »*

Objectif : Utiliser la structure « `pair` » et les conteneurs « `list` » et « `map` » **de la bibliothèque standard C++.**

Une liste simple

Sachant que dans la bibliothèque standard C++ :

- La classe list est déclarée comme suit :
```
template <class T, class A = allocator <T> >
class list;
```
- La classe list comporte les déclarations :
```
typedef T value_type;          // équivaut à:
typedef value_type* iterator;  // typedef T* iterator;
```
- La méthode void push_back (const value_type&) ajoute un nouvel élément en fin de liste
- Le type iterator défini dans la classe est utilisé pour itérer sur une liste
- Les méthodes iterator begin() et iterator end() retournent respectivement un pointeur sur le début de la liste et sur l'emplacement qui suit le dernier élément de la liste

Travail à faire

1. Représenter en C++, les caractéristiques de la classe list énoncées ci-dessus

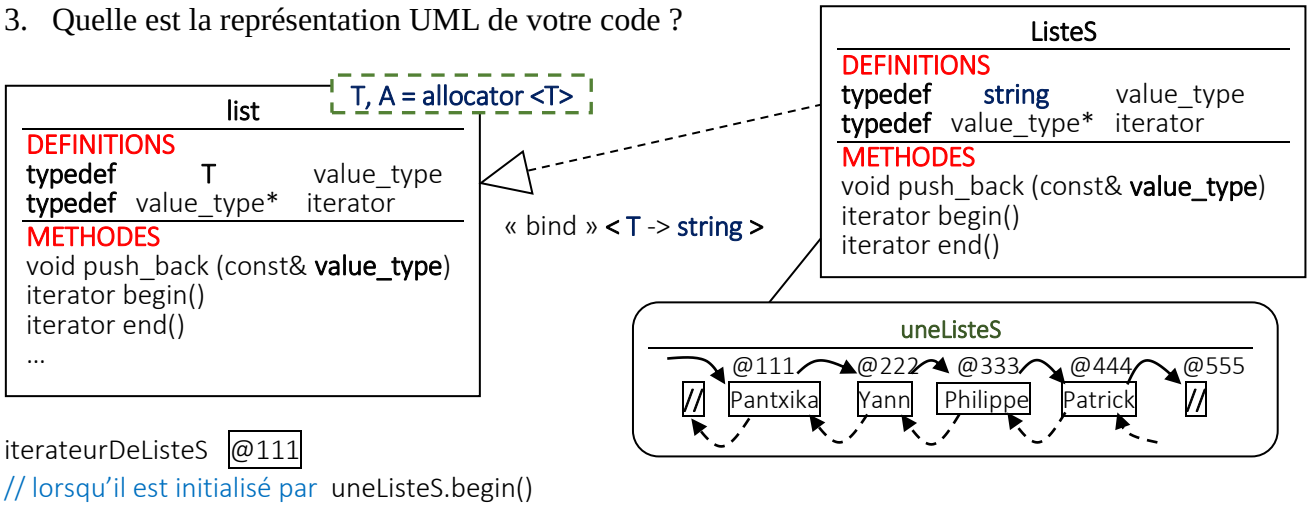
```
template <class T, class A = allocator <T> >
class list {
        public:
                // Définition de types
                typedef    T          value_type;          // Equivaut à :
                typedef value_type* iterator;             // typedef   T*  iterator;

                // Peuplement
                void push_back(const value_type&)         // Ajoute un nouvel élément en fin de liste

                // Accès
                iterator begin();
                iterator end();
}
```

2. En vous inspirant des pages 5, 6, 7 et 8 du support de cours TemplateC++ situé sur eLearn, écrire une procédure listeSimple() dans un fichier main.cpp qui :
   a) Sur le modèle de classe générique list, crée une classe ListeS, liste de string
   b) Crée un objet uneListeS, instance de ListeS
   c) Alimente la liste avec les valeurs "Pantxika", "Yann", "Philippe" et "Patrick"
   d) Crée un iterateur pour accéder aux éléments de uneListeS
   e) Initialise iterateur au premier élément de uneListeS
   f) Fait un parcours avant et complet de la liste et affiche chaque élément

Note : Insérer l'instruction #include <list> dans le main.cpp pour pouvoir utiliser la classe générique list définie dans la bibliothèque standard C++.

```
void listeSimple () {

        // Définit une classe ListeS, pour instancier des listes de strings
        typedef list<string> ListeS;

        // Crée un objet uneListeS, instance de ListeS
        ListeS uneListeS;

        // Alimente la liste avec "Pantxika", "Yann", "Philippe" et "Patrick"
        uneListeS.push_back (ListeS::value_type ("Pantxika"));
        uneListeS.push_back (ListeS::value_type ("Yann"));
        uneListeS.push_back (ListeS::value_type ("Philippe"));
        uneListeS.push_back (ListeS::value_type ("Patrick"));

        // Crée un itérateur pour parcourir les éléments qui composent les listes
        // qui sont instances de la classe ListeS
        ListeS::iterator iterateurDeListeS;     // Peut contenir d'adresse d'un élément qui
                                                // compose une liste instance de ListeS
        // Initialise l'itérateur au premier élément de l'instance uneListeS
        iterateurDeListeS = uneListeS.begin();

        // Parcours complet de la liste uneListeS et affiche chaque élément
        while (iterateurDeListeS != uneListeS.end()) {
                cout << *iterateurDeListeS << endl;
                iterateurDeListeS++;            // Accède à l'élément suivant qui compose la liste
        }
}
```

3. Quelle est la représentation UML de votre code ?



iterateurDeListeS @111
// lorsqu'il est initialisé par uneListeS.begin()

Rappel : les éléments de list sont doublement chaînés (cf. begin()⟶end() et rbegin() - - ▸ rend()).

4. Appeler la procédure listeSimple() depuis la fonction principale main().

Une liste de paires

Sachant que dans la bibliothèque standard C++ :

- La classe pair est déclarée comme suit :
  ```
  template     <class T1, class T2>
  class pair;
  ```
- La classe pair comporte les déclarations
  ```
  typedef    T1  first_type;
  typedef    T2  second_type;
  ```
- Le constructeur pair (const first_type&, const second_type&) permet d'initialiser un objet
- Les attributs first et second représentent la 1ère et 2ème composante de la paire

Travail à faire

5. Représenter en C++, les caractéristiques de pair énoncées ci-dessus

```
template <class T1, class T2>
class pair {
        public:
                // Définitions de types
                typedef   T1    first_type;
                typedef   T2    second_type;

                // Attributs
                first_type      first;          // 1ère composante de la paire
                second_type     second;         // 2ème composante de la paire

                // Constructeur
                pair (const first_type&, const second_type&);
}
```

6. Ecrire un exemple qui crée et initialise une paire de chaines de caractères.

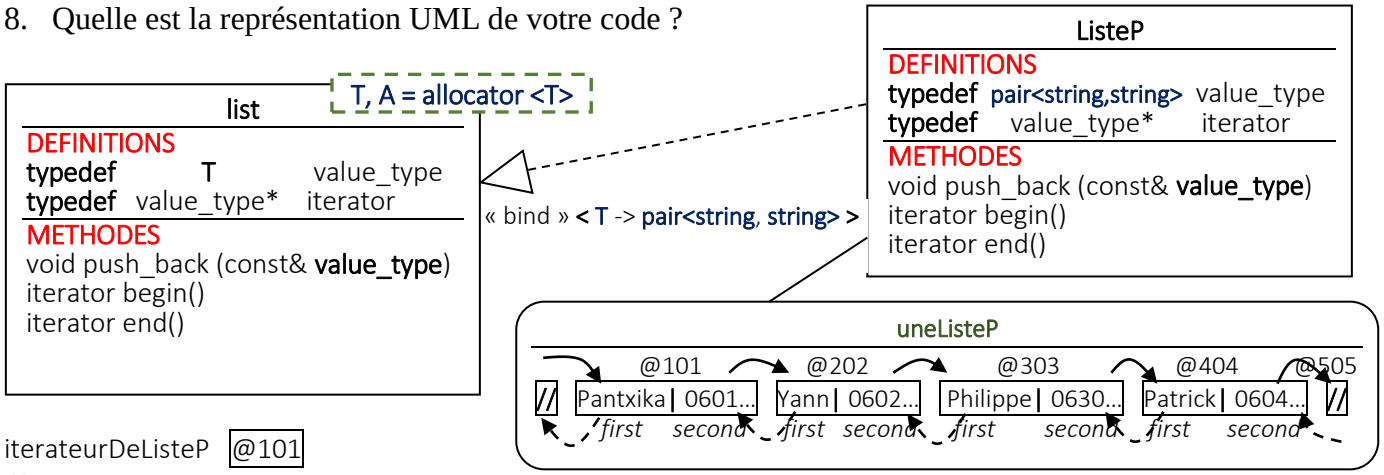
```
// Définit un type PairStringString
typedef pair    <string, string>    PairStringString;
// Déclaration de unePaire, instance de PairStringString
PairStringString    unePaire;
// Initialise les 2 parties de unePaire
unePaire.first = "partie 1" ;
unePaire.second = "partie 2";
```

```
/* version courte ci-dessous */

// Déclare et initialise autrePaire grâce au constructeur
pair <string, string>    autrePaire ("partie A", "partie B");
```

7. Ecrire une procédure listePaires() dans le fichier main.cpp qui :
   a) Construit une classe ListeP, liste de paires de string
   b) Crée un objet uneListeP, instance de la ListeP
   c) Alimente la liste avec les valeurs ("Pantxika", "06.01.01.01.01"), ("Yann", "06.02.02.02.02"), ("Philippe", "06.03.03.03.03") et ("Patrick", "06.04.04.04.04")
   d) Crée un iterateur pour accéder aux éléments de uneListeP
   e) Initialise iterateur au premier élément de uneListeP
   f) Fait un parcours avant de la liste et affiche les composants des éléments

```
void listePaires () {
        // a. Définit le type PairStringString
        typedef pair <string, string> PairStringString;

        // b. Définit une classe ListeP, liste de paires de strings
        typedef list <PairStringString> ListeP;

        // Ou bien a. & b. d'un seul coup : définition d'une classe ListeP, liste de paires de strings
        /*  typedef list <pair <string, string> > ListeP;  */

        // Crée un objet uneListeP, instance de la ListeP
        ListeP uneListeP;

        // Alimente la liste avec des valeurs
        uneListeP.push_back (ListeP::value_type ("Pantxika ", "06.01.01.01.01"));
        uneListeP.push_back (ListeP::value_type ("Yann", "06.02.02.02.02"));
        uneListeP.push_back (ListeP::value_type ("Philippe", "06.03.03.03.03"));
        uneListeP.push_back (ListeP::value_type ("Patrick", "06.04.04.04.04"))

        // Crée un itérateur pour parcourir les éléments qui composent les instances de ListeP
        ListeP::iterator        iterateurDeListeP;

        // Initialise l'iterateur au premier element de uneListeP, instance de ListeP
        iterateurDeListeP = uneListeP.begin();

        // Parcours complet de la liste uneListeP et affiche les composants de chacun de ses elements
        while (iterateurDeListeP != uneListeP.end()) {
                cout << iterateurDeListeP->first << " - " << iterateurDeListeP->second << endl;
                iterateurDeListeP++;      // Accède à l'élément suivant
        }
}
```

8. Quelle est la représentation UML de votre code ?



iterateurDeListeP @101

// une fois initialisé avec le résultat de la fonction uneListeP.begin()

9. Appeler la procédure listePaires() depuis la fonction principale main().

Un map

Sachant que dans la bibliothèque standard C++ :

- La classe map est déclarée dans le #include <map> comme suit
  template <class Key, class T, class Cmp = less <Key>, class A = allocator <T> >
  class map;
- La classe map comporte les déclarations :
  typedef Key key_type;
  typedef T mapped_type;
  typedef pair <const key_type, mapped_type> value_type;
  typedef value_type* iterator; // type "pointeur sur élément"
- La méthode pair <iterator, bool> insert (value_type& x) insère l'élément x dans le map. Si l'insertion se fait correctement, retourne l'adresse ou est inséré x et true. Sinon retourne l'adresse ou est positionné l'élément de même clef que x et false.
- La méthode iterator find(const key_type& k) retourne un pointeur sur un élément de clé k, ou bien la valeur de end() si la clef k n'existe pas dans le map.

Travail à faire

10. Représenter en C++, les caractéristiques de map énoncées ci-dessus

```
template <class Key, class T, class Cmp = less <Key>, class A = allocator <T> >
class map {
        public:
                // Définitions de types
                typedef Key key_type;
                typedef T mapped_type; // A partir de ce point Key et T sont non utilisés
                typedef pair <const key_type, mapped_type> value_type;
                typedef value_type* iterator;

                // Peuplement
                pair <iterator, bool> insert (value_type& x);   // Ajoute un nouvel élément

                // Accès
                iterator find(const key_type& k);
}
```

11. Ecrire une procédure leMap() dans le fichier main.cpp qui :
    a) Définit une classe Annuaire, du type map de string ayant string comme clef
    b) Crée un objet unAnnuaire, instance de Annuaire
    c) Crée resultatInsert une paire de valeurs qui récupère le résultat d'une insertion
    d) Alimente l'annuaire avec ("Pantxika", "06.01.01.01.01"), teste le résultat retourné et affiche le message "Insertion BIEN réalisée" ou bien le message "Insertion MAL réalisée".

e) Duplique la section de code du c) pour constater que la seconde insertion n'aboutit pas puisque dans un map, la clef est unique.

f) Insère ("Yann","06.02.02.02.02"), ("Philippe", "06.03.03.03.03") et ("Patrick", "06.04.04.04.04")

g) Crée un iterateur pour accéder aux éléments de unAnnuaire

h) Initialise l'iterateur au premier élément de unAnnuaire

i) Fait un parcours avant du map et affiche le second composant de chaque élément accédé

j) Cherche (cf. positionne) l'iterateur sur l'élément de unAnnuaire ayant "Philippe" comme clef

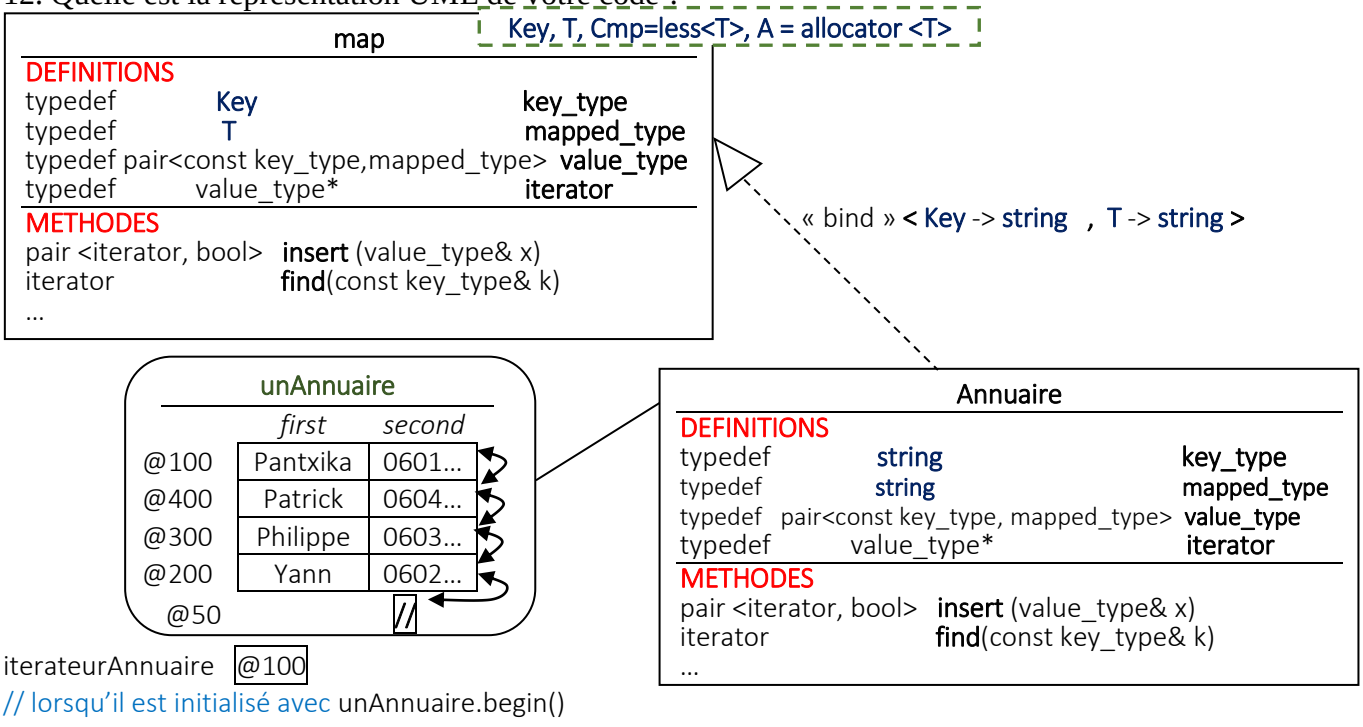
k) Analyse la valeur retournée et affiche le numéro de téléphone, ou bien le message "Philippe" clef inconnue.

Note : Insérer l'instruction #include <map> dans le main.cpp pour pouvoir utiliser la classe générique map définie dans la bibliothèque standard C++.

```
void leMap () {
/// DELCARATIONS
        // Définit une classe Annuaire, une map de string avec string en clef
        typedef    map <string, string>    Annuaire;
        // Crée un objet unAnnuaire, instance de Annuaire
        Annuaire                           unAnnuaire;
        // Etablit le type IterateurAnnuaire
        typedef      Annuaire::iterator      IterateurAnnuaire;
        // Crée resultatInsert une paire de valeurs qui récupère le résultat d'une insertion
        pair <IterateurAnnuaire, bool>      resultatInsert;  // On aurait pu passer par un typedef
        // La seconde valeur vaut true si l'insertion est fructueuse et false sinon
        // En cas d'insertion fructueuse, la première valeur contient l'adresse de l'élément inséré
        // En cas d'insertion infructueuse, la première valeur contient l'adresse de l'élément ayant
        // la même clef
/// TRAITEMENTS
        // Alimente l'annuaire avec "Pantxika", "06.01.01.01.01" et teste le résultat
        resultatInsert=unAnnuaire.insert(Annuaire::value_type("Pantxika", "06.01.01.01.01"));
        if (resultatInsert.second == true)              // Si insertion fructueuse
                cout << "Insertion " << resultatInsert.first -> first
                << " BIEN realisee" << endl;
        else                                            // Sinon
                cout << "Insertion MAL realisee" << endl;
        // Réalimente l'annuaire avec "Pantxika", "06.01.01.01.01" et teste le résultat
        resultatInsert=unAnnuaire.insert(Annuaire::value_type("Pantxika", "06.01.01.01.01"));
        if (resultatInsert.second == true)              // Si insertion infructueuse
                cout << "Insertion " << resultatInsert.first -> first
                << " BIEN realisee" << endl;
        else                                            // Sinon
                cout << "Insertion MAL realisee" << endl;
```

```
        // Alimente unAnnuaire de trois autres tuples
        unAnnuaire.insert (Annuaire::value_type ("Yann", "06.02.02.02.02"));
        unAnnuaire.insert (Annuaire::value_type ("Philippe", "06.03.03.03.03"));
        unAnnuaire.insert (Annuaire::value_type ("Patrick", "06.04.04.04.04"));

        // Crée un iterateur pour parcourir les objets de Annuaire
        IterateurAnnuaire    unInterateur;

        // Initialise l'iterateur au premier élément de unAnnuaire
        unInterateur = unAnnuaire.begin ();

        // Parcours complet de la liste et affiche le second composant de chaque élément
        while (unInterateur != unAnnuaire.end()) {
                cout << unInterateur -> second << endl;
                unInterateur ++;        // Accès à l'élément suivant
        }
        // La méthode find() retourne un itérateur sur la paire de clefRecherchee ou bien sur end()
        string clefRecherchee = "Philippe";
        unInterateur = unAnnuaire.find (clefRecherchee);

        // Affiche le numéro de téléphone ou bien : Philippe - clef inconnue
        if (unInterateur != unAnnuaire.end())
                cout << clefRecherchee << "  - telephone : " << unInterateur->second << endl;
        else    cout << clefRecherchee  << " clef inconnue" << endl;
}
```

12. Quelle est la représentation UML de votre code ?



iterateurAnnuaire @100
// lorsqu'il est initialisé avec unAnnuaire.begin()

Note : *Une différence notable par rapport au list de paires, est que le map range les éléments dans l'ordre de la clef Key, au fur et mesure des insertions (sous forme d'arbre binaire trié – mais ici représenté sous forme de table). De plus le map offre une méthode de recherche find sur la clef Key.*

13. Appeler la procédure leMap() depuis la fonction principale main().