Correction TD 4 :

Généricité paramétrique : Dictionnaire

L3, C++

FLIN606

Assoc.h

```
#ifndef  Assoc_h
  #define   Assoc_h


using namespace std;
#include<iostream>

template<typename TypeCle, typename TypeValeur>
class Assoc{
private:
  TypeCle cle; TypeValeur valeur;
public:
  Assoc();
  Assoc(TypeCle, TypeValeur);
  virtual ~Assoc ();
  virtual TypeCle getCle()const;
  virtual void setCle(TypeCle);
  virtual TypeValeur getValeur()const;
  virtual void setValeur(TypeValeur);
  virtual void affiche(ostream&)const;
};

template<typename TypeCle, typename TypeValeur>
ostream& operator<<(ostream&, const Assoc<TypeCle,TypeValeur>&);

#endif
```

Assoc.cc

```
#include "Assoc.h"

template<typename TypeCle, typename TypeValeur>
Assoc<TypeCle,TypeValeur>::Assoc()
 {}

template<typename TypeCle, typename TypeValeur>
Assoc<TypeCle,TypeValeur>::Assoc(TypeCle c, TypeValeur v)
:cle(c), valeur(v)
 {}

template<typename TypeCle, typename TypeValeur>
Assoc<TypeCle,TypeValeur>::~Assoc ()
 {}

template<typename TypeCle, typename TypeValeur>
TypeCle Assoc<TypeCle,TypeValeur>::getCle()const
 {return cle;}

template<typename TypeCle, typename TypeValeur>
void Assoc<TypeCle,TypeValeur>::setCle(TypeCle c)
 {cle=c;}

template<typename TypeCle, typename TypeValeur>
TypeValeur Assoc<TypeCle,TypeValeur>::getValeur()const
 {return valeur;}

template<typename TypeCle, typename TypeValeur>
void Assoc<TypeCle,TypeValeur>::setValeur(TypeValeur v)
 {valeur=v;}

template<typename TypeCle, typename TypeValeur>
void Assoc<TypeCle,TypeValeur>::affiche(ostream &os) const
  {os <<getCle() << ", " <<getValeur();}


template<typename TypeCle, typename TypeValeur>
ostream& operator<<(ostream& os, const Assoc<TypeCle,TypeValeur>& a)
  {a.affiche(os); return os;}
```

AssocStringInt.cc

```
#include "Assoc.cc"
#include <string>

template class Assoc<string, int>;

template ostream& operator<<(ostream&, const Assoc<string, int>&);
```

Dico.h

```
#ifndef Dico_h
#define Dico_h
using namespace std;
#include<iostream>

#include "Assoc.h"
template<typename TypeCle, typename TypeValeur>
class Dico
{
  private :

  Assoc<TypeCle, TypeValeur> * tabAssoc;
  int nbAssoc;
  int capacite;

  void CherchCl(const TypeCle& cl, int& i, int& res)const;

  static TypeCle cleDefaut;
  static TypeValeur valeurDefaut;

  public :

  static int hash(TypeCle, int);
  Dico();
  Dico(const Dico & D);
  virtual ~Dico();
  virtual void put (const TypeCle &, const TypeValeur &);
  virtual TypeValeur get(const TypeCle &)const;
  virtual bool estVide()const;
  virtual bool contient(const TypeCle& C)const;
  virtual int taille()const;
  virtual void  affiche (ostream&)const;
  virtual Dico& operator=(const Dico & D);

};

template<typename TypeCle, typename TypeValeur>
ostream& operator<<(ostream& , const Dico<TypeCle, TypeValeur>& );

#endif
```

Dico.cc

```
#include<string>
#include "Dico.h"

/*
les attributs statiques cleDefaut et valeurDefaut, et la methode statique
hash
ne peuvent pas être définis dans le cas general: il faudra le faire
dans le fichier qui definit une "vraie" classe (ex:DicoStringInt.cc).
Remarque: ce n'est pas parce que ce sont des attributs ou des méthodes
statiques
 qu'ils ne peuvent pas etre definis. C'est parce qu'ils utilisent les types
paramétrés et qu'on a besoin des types effectifs pour pouvoir écrire leur
code.
Il peut exister des attributs ou methodes statiques qui se definissent sans
probleme dans le cas general,et des attributs ou méthodes non statiquee qui
ne
peuvent pas être définis.
*/

/******constructeur sans
parametres*************************************/
template<typename TypeCle, typename TypeValeur>
Dico<TypeCle,TypeValeur>::Dico()
 {tabAssoc = new Assoc<TypeCle, TypeValeur>[10];
  for (int i=0;i<10;i++)
   {tabAssoc[i].setCle(cleDefaut);tabAssoc[i].setValeur(valeurDefaut);}
  nbAssoc = 0;capacite = 10;}

/******constructeur par copie*****************************************/
template<typename TypeCle, typename TypeValeur>
Dico<TypeCle,TypeValeur>::Dico(const Dico<TypeCle, TypeValeur> & D)
 { capacite = D.capacite;nbAssoc = D.nbAssoc;
  tabAssoc = new Assoc<TypeCle, TypeValeur>[capacite];
  for (int i=0; i<capacite; i++) tabAssoc[i] = D.tabAssoc[i];}

/******destructeur*****************************************************/
template<typename TypeCle, typename TypeValeur>
Dico<TypeCle,TypeValeur>::~Dico()
 {delete[] tabAssoc;}

/*******methode privee*************************************************/
template<typename TypeCle, typename TypeValeur>
void Dico<TypeCle,TypeValeur>::CherchCl
               (const TypeCle& cl, int& i, int& res)const
   /* cherche la cle cl dans le dictionnaire :
    si cl est presente: renvoie res=1, i indice de la case de cl dans T;
    si cl est absente et le dictionnaire non plein: renvoie res=0, i indice
      de case possible pour cl dans T;
    si cl est absente et le dictionnaire plein: renvoie res=2, i non
     significatif. */
 {int j=hash(cl,capacite);
  i=j; res=0;
  while (tabAssoc[i].getCle()!=cleDefaut && tabAssoc[i].getCle()!=cl
          && res!=2 )
    {i++;if (i==capacite) i=0; if (i==j) res=2;}
  if (tabAssoc[i].getCle()==cl ) res=1;
 }
```

```
/*******methodes publiques*********************************************/

//------------put-----------------------------------------------------
template<typename TypeCle, typename TypeValeur>
void Dico<TypeCle,TypeValeur>::put (const TypeCle& c, const TypeValeur& v)
 {int i,res;CherchCl (c,i,res);
  switch(res)
  {
    case 0://cle absente qui peut etre placee a l'indice i
     {tabAssoc[i].setCle(c);tabAssoc[i].setValeur(v);nbAssoc++;return;}

    case 1: //cle presente:on change la valeur
    {tabAssoc[i].setValeur(v);return;}

   case 2: //cle absente, et c'est plein; il faut agrandir.
     {
     //tableau auxiliaire, rempli en copiant le tableau actuel du Dico
        Assoc<TypeCle, TypeValeur> * tabAux=new Assoc<TypeCle,
TypeValeur>[capacite];
        for (int j=0; j<capacite; j++) tabAux[j] = tabAssoc[j];
     //effacement du tableau actuel du Dico
        delete []tabAssoc;
     //creation et initialisation d'un nouvel espace plus grand
     //pour le tableau du Dico
        capacite=capacite+5;nbAssoc=0;
        tabAssoc=new Assoc<TypeCle, TypeValeur>[capacite];
        for (int j=0;j<capacite;j++)

{tabAssoc[j].setCle(cleDefaut);tabAssoc[j].setValeur(valeurDefaut);}
     //remplissage de ce nouveau tableau à partir des elts
     //du tableau auxiliaire, en utilisant le hachage
        for (int j=0;j<capacite-5;j++)
          {CherchCl(tabAux[j].getCle(),i,res);
           tabAssoc[i].setCle(tabAux[j].getCle());
           tabAssoc[i].setValeur(tabAux[j].getValeur());
            nbAssoc++;}
     //maintenant il y a de la place: ou caser la nouvelle association?
        CherchCl(c,i,res);//cet appel renvoie toujours 0 pour res
        tabAssoc[i].setCle(c);tabAssoc[i].setValeur(v);nbAssoc++;
     //on efface les elements du tableau auxiliaire
       delete[]tabAux;
     }
  }
 }

//-----------get-----------------------------------------------------
template<typename TypeCle, typename TypeValeur>
TypeValeur Dico<TypeCle,TypeValeur>::get(const TypeCle & c)const
 {int i,res;CherchCl(c,i,res);
  if (res==1) return tabAssoc[i].getValeur();
   else return valeurDefaut;//ce serait mieux de faire une exception
 }

//-----------estVide-------------------------------------------------
template<typename TypeCle, typename TypeValeur>
bool Dico<TypeCle,TypeValeur>::estVide()const
  {return  nbAssoc==0;}

//-----------contient-----------------------------------------------
template<typename TypeCle, typename TypeValeur>
```

```
bool Dico<TypeCle,TypeValeur>::contient(const TypeCle& C)const
 {int i,res; CherchCl(C,i,res); return (res==1);}

//----------taille-------------------------------------------------
template<typename TypeCle, typename TypeValeur>
int Dico<TypeCle,TypeValeur>::taille()const
 {return nbAssoc;}

//----------affiche------------------------------------------------
template <typename TypeCle, typename TypeValeur>
void Dico<TypeCle,TypeValeur>::affiche(ostream& os)const
 {if (estVide()) cout<<"Dico vide"<<endl;
  else
   {cout << "\nIndice  Cle  \t\t\tValeur\n"<<endl;
    for (int i=0; i<capacite; i++)
      {if (tabAssoc[i].getCle() != cleDefaut)
        {os << i <<"\t"<< tabAssoc[i].getCle();
         for (int j=0;j<24-tabAssoc[i].getCle().length();j++) os <<" ";
         os << tabAssoc[i].getValeur() <<endl;
        }
      }
   }
}

//----------surcharge de l'operateur =-------------------------------
template<typename TypeCle, typename TypeValeur>
Dico<TypeCle,TypeValeur>&
Dico<TypeCle,TypeValeur>::operator=(const Dico<TypeCle, TypeValeur> & D)
 {if (this !=&D)
  {delete[] tabAssoc;
   capacite = D.capacite;
   nbAssoc = D.nbAssoc;
   tabAssoc = new Assoc<TypeCle, TypeValeur>[capacite];
   for (int i=0; i<capacite; i++) tabAssoc[i] = D.tabAssoc[i];
  }
  return *this;}

/*******fonctions (extérieures à la classe) ***************************/

//----------surcharge de l'operateur <<-----------------------------
template <typename TypeCle, typename TypeValeur>
ostream& operator<<(ostream& os, const Dico<TypeCle, TypeValeur>& D)
 {D.affiche(os); return os;}
```

Dico2.h

```
#ifndef Dico2_h
#define Dico2_h
using namespace std;
#include<iostream>

#include "Assoc.h"
template<typename TypeCle, typename TypeValeur>
class Dico
{
  private :

  Assoc<TypeCle, TypeValeur> * tabAssoc;
  int nbAssoc;
  int capacite;
  TypeCle cleDefaut;
  TypeValeur valeurDefaut;

  void CherchCl(const TypeCle& cl, int& i, int& res)const;


  public :

  static int hash(TypeCle, int);
  Dico();
  Dico(const Dico & D);
  virtual ~Dico();
  virtual void put (const TypeCle &, const TypeValeur &);
  virtual TypeValeur get(const TypeCle &)const;
  virtual bool estVide()const;
  virtual bool contient(const TypeCle& C)const;
  virtual int taille()const;
  virtual void  affiche (ostream&)const;
  virtual Dico& operator=(const Dico & D);

};

template<typename TypeCle, typename TypeValeur>
ostream& operator<<(ostream& , const Dico<TypeCle, TypeValeur>& );

#endif
```

Dico2.cc

```
#include<string>
#include "Dico2.h"

/*
les attributs cleDefaut et valeurDefaut, et la methode statique hash
ne peuvent pas être définis dans le cas general: il faudra le faire
dans le fichier qui definit une "vraie" classe (ex:DicoStringInt.cc).

*/

/******constructeur sans
parametres*************************************/
template<typename TypeCle, typename TypeValeur>
Dico<TypeCle,TypeValeur>::Dico()
{//a ecrire dans le fichier d'instanciation;ici cleDefaut
 //et valeurDefaut ne sont pas initialisees
}

/******constructeur par copie*************************************************/
template<typename TypeCle, typename TypeValeur>
Dico<TypeCle,TypeValeur>::Dico(const Dico<TypeCle, TypeValeur> & D)
 { capacite = D.capacite;nbAssoc = D.nbAssoc;
  tabAssoc = new Assoc<TypeCle, TypeValeur>[capacite];
  for (int i=0; i<capacite; i++) tabAssoc[i] = D.tabAssoc[i];}

/******destructeur*************************************************************/
template<typename TypeCle, typename TypeValeur>
Dico<TypeCle,TypeValeur>::~Dico()
 {delete[] tabAssoc;}

/*******methode privee*********************************************************/
template<typename TypeCle, typename TypeValeur>
void Dico<TypeCle,TypeValeur>::CherchCl
               (const TypeCle& cl, int& i, int& res)const
   /* cherche la cle cl dans le dictionnaire :
    si cl est presente: renvoie res=1, i indice de la case de cl dans T;
    si cl est absente et le dictionnaire non plein: renvoie res=0, i indice
      de case possible pour cl dans T;
    si cl est absente et le dictionnaire plein: renvoie res=2, i non
     significatif. */
 {int j=hash(cl,capacite);
  i=j; res=0;
  while (tabAssoc[i].getCle()!=cleDefaut && tabAssoc[i].getCle()!=cl
          && res!=2 )
    {i++;if (i==capacite) i=0; if (i==j) res=2;}
  if (tabAssoc[i].getCle()==cl ) res=1;
 }


/*******methodes publiques*****************************************/

//------------put-----------------------------------------------------
template<typename TypeCle, typename TypeValeur>
void Dico<TypeCle,TypeValeur>::put (const TypeCle& c, const TypeValeur& v)
 {int i,res;CherchCl (c,i,res);
  switch(res)
  {
    case 0://cle absente qui peut etre placee a l'indice i
```

```
     {tabAssoc[i].setCle(c);tabAssoc[i].setValeur(v);nbAssoc++;return;}

    case 1: //cle presente:on change la valeur
    {tabAssoc[i].setValeur(v);return;}

   case 2: //cle absente, et c'est plein; il faut agrandir.
     {
     //tableau auxiliaire, rempli en copiant le tableau actuel du Dico
        Assoc<TypeCle, TypeValeur> * tabAux=new Assoc<TypeCle,
TypeValeur>[capacite];
        for (int j=0; j<capacite; j++) tabAux[j] = tabAssoc[j];
     //effacement du tableau actuel du Dico
        delete []tabAssoc;
     //creation et initialisation d'un nouvel espace plus grand
     //pour le tableau du Dico
        capacite=capacite+5;nbAssoc=0;
        tabAssoc=new Assoc<TypeCle, TypeValeur>[capacite];
        for (int j=0;j<capacite;j++)

{tabAssoc[j].setCle(cleDefaut);tabAssoc[j].setValeur(valeurDefaut);}
     //remplissage de ce nouveau tableau à partir des elts
     //du tableau auxiliaire, en utilisant le hachage
        for (int j=0;j<capacite-5;j++)
          {CherchCl(tabAux[j].getCle(),i,res);
           tabAssoc[i].setCle(tabAux[j].getCle());
           tabAssoc[i].setValeur(tabAux[j].getValeur());
            nbAssoc++;}
     //maintenant il y a de la place: ou caser la nouvelle association?
        CherchCl(c,i,res);//cet appel renvoie toujours 0 pour res
        tabAssoc[i].setCle(c);tabAssoc[i].setValeur(v);nbAssoc++;
     //on efface les elements du tableau auxiliaire
       delete[]tabAux;
     }
  }
 }

//-----------get-------------------------------------------------------
template<typename TypeCle, typename TypeValeur>
TypeValeur Dico<TypeCle,TypeValeur>::get(const TypeCle & c)const
 {int i,res;CherchCl(c,i,res);
  if (res==1) return tabAssoc[i].getValeur();
   else return valeurDefaut;//ce serait mieux de faire une exception
 }

//-----------estVide---------------------------------------------------
template<typename TypeCle, typename TypeValeur>
bool Dico<TypeCle,TypeValeur>::estVide()const
  {return  nbAssoc==0;}

//-----------contient--------------------------------------------------
template<typename TypeCle, typename TypeValeur>
bool Dico<TypeCle,TypeValeur>::contient(const TypeCle& C)const
 {int i,res; CherchCl(C,i,res); return (res==1);}

//-----------taille----------------------------------------------------
template<typename TypeCle, typename TypeValeur>
int Dico<TypeCle,TypeValeur>::taille()const
 {return nbAssoc;}

//-----------affiche---------------------------------------------------
template <typename TypeCle, typename TypeValeur>
```

```
void Dico<TypeCle,TypeValeur>::affiche(ostream& os)const
 {if (estVide()) cout<<"Dico vide"<<endl;
  else
   {cout << "\nIndice  Cle  \t\t\tValeur\n"<<endl;
    for (int i=0; i<capacite; i++)
      {if (tabAssoc[i].getCle() != cleDefaut)
        {os << i <<"\t"<< tabAssoc[i].getCle();
         for (int j=0;j<24-tabAssoc[i].getCle().length();j++) os <<" ";
         os << tabAssoc[i].getValeur() <<endl;
        }
      }
   }
}

//----------surcharge de l'operateur =------------------------------
template<typename TypeCle, typename TypeValeur>
Dico<TypeCle,TypeValeur>&
Dico<TypeCle,TypeValeur>::operator=(const Dico<TypeCle, TypeValeur> & D)
 {if (this !=&D)
  {delete[] tabAssoc;
   capacite = D.capacite;
   nbAssoc = D.nbAssoc;
   tabAssoc = new Assoc<TypeCle, TypeValeur>[capacite];
   for (int i=0; i<capacite; i++) tabAssoc[i] = D.tabAssoc[i];
  }
  return *this;}

/*******fonctions (extérieures à la classe) ***************************/

//----------surcharge de l'operateur <<-----------------------------
template <typename TypeCle, typename TypeValeur>
ostream& operator<<(ostream& os, const Dico<TypeCle, TypeValeur>& D)
 {D.affiche(os); return os;}
```

DicoStringInt.cc

```
#include "Dico.cc"
#include <string>

//def de hash : a placer avant l'instanciation  de la classe!
template<> int Dico<string,int>::hash(string C, int tailleTab)
{int i=0;
 for (int j=0; j<C.length(); j++)i+=(j+1)*C[j];
 return (i % tailleTab);
 }

template<> string Dico<string,int>::cleDefaut="rien";
template<> int Dico<string,int>::valeurDefaut=0;

template class Dico<string, int>;

template ostream& operator<<(ostream& , const Dico<string, int>& );

/*
ce fichier suppose que Dico.h contient les declarations:

  static TypeCle cleDefaut; //valeur par defaut pour la cle
  static TypeValeur valeurDefaut; //valeur par defaut pour la valeur

  static int hash(TypeCle, int); //methode de hachage

ces deux variables et cette méthode sont necessaires pour ecrire le code
des diverses methodes dans Dico.cc, mais on ne peut fixer les valeurs
de ces variables et le contenu de cette methode que lorsqu'on connait
les types effectifs de cle et valeur.
On n'a donc rien dans Dico.cc qui correspond a leurs declarations dans
Dico.h, mais on doit le faire dans DicoStringInt.cc avant d'ecrire la ligne
qui cree vraiment la classe:
  template class Dico<string,int>;
*/
```

DicoStringInt2.cc

```
#include "Dico2.cc"
#include <string>

using namespace std;
/**redef du constructeur sans
parametres************************************/

template<> Dico<string,int>::Dico()
 {
  cleDefaut="rien";valeurDefaut=0;
  tabAssoc = new Assoc<string,int>[10];
  for (int i=0;i<10;i++)
   {tabAssoc[i].setCle(cleDefaut);tabAssoc[i].setValeur(valeurDefaut);}
  nbAssoc = 0;capacite = 10;
  }

//def de hash : a placer avant l'instanciation  de la classe!
 template<> int Dico<string,int>::hash(string C, int tailleTab)
{int i=0;
 for (int j=0; j<C.length(); j++)i+=(j+1)*C[j];
 return (i % tailleTab);
 }

template class Dico<string, int>;

template ostream& operator<<(ostream& , const Dico<string, int>& );

/*
ce fichier suppose que Dico.h contient les declarations:

  TypeCle cleDefaut; //valeur par defaut pour la cle
  TypeValeur valeurDefaut; //valeur par defaut pour la valeur

  static int hash(TypeCle, int); //methode de hachage

ces deux variables et cette méthode sont necessaires pour ecrire le code
des diverses methodes dans Dico.cc, mais on ne peut fixer les valeurs
de ces variables et le contenu de cette methode que lorsqu'on connait
les types effectifs de cle et valeur.
On n'a donc rien dans Dico2.cc qui correspond a leurs declarations dans
Dico2.h, mais on doit le faire dans DicoStringInt2.cc avant d'ecrire la
ligne
qui cree vraiment la classe:
  template class Dico<string,int>;
*/
```

DicoMainCorrig.cc

```
using namespace std;
#include<iostream>
#include<string>
#include"Dico.h"

void compteMots()
{
// Compte le nombre de mots d'un texte lu sur l'entree standard
/* rappels:
 1)quand on lit une string, le premier blanc, ou la premiere tabulation,
  ou le premier saut de ligne rencontré la termine.
 2) cin>>Mot renvoie cin (cf signature de operator>>); donc
   while (cin>>Mot) teste l'etat de cin pour continuer (et continue si on
   n'a pas eu d'erreur et si on n'est pas en eof)
 3)eof :si cin est redirigé vers un fichier, c'est la fin du fichier;
   s'il n'est pas redirigé, taper Ctrl-D
*/
 Dico<string,int> D;
 string Mot;
 while (cin >> Mot)
     {if (D.contient(Mot)){int v=D.get(Mot);v++;D.put(Mot,v);}
       else D.put(Mot,1);
     }
 cout << D;
}//compteMots

int main()
{

  // Test de la fonction de hachage en longueur 10
  //----------------------------------------------
  // permet de voir la valeur calculee par la fonction
  cout << "\nhash(abricot, 10) = "
       << Dico<string,int>::hash("abricot", 10) <<endl;

 //declaration et remplissage d'un dictionnaire
 //---------------------------------------------
  Dico<string,int> D;  //par defaut ,taille 10
  cout << "\nNbAssociations =" << D.taille() << "\n" << D <<endl;

  //permet de voir dans quelles cases ca va reellement aller
  D.put("abricot", 11);
  D.put("amande", 22);
  D.put("ananas",33);
  D.put("pomme", 44);
  cout << "\nNbAssociations =" << D.taille() << "\n" << D <<endl;

  D.put("prune",55);
  D.put("griotte", 66);
  D.put("poire", 77);
  D.put("orange",88);
  D.put("citron",99);
  D.put("mangue",100);
  cout << "\nNbAssociations =" << D.taille() << "\n" << D <<endl;
  // la le dico est plein
```

```
  //on ajoute une assoc: le dico allonge de 5=> taille 15
  D.put("papaye",11);
  cout << "\nNbAssociations =" << D.taille() << "\n" << D <<endl;


//Comptage de mots
//----------------
  cout << "Comptage des mots dans un texte \n";
  compteMots();
}

/*
resultats:
---------
hash(abricot, 10) = 8


NbAssociations =0
Dico vide


NbAssociations =4

Indice  Cle                   Valeur

2       amande                 22
3       ananas                 33
4       pomme                  44
8       abricot                11


NbAssociations =10

Indice  Cle                   Valeur

0       poire                  77
1       orange                 88
2       amande                 22
3       ananas                 33
4       pomme                  44
5       griotte                66
6       prune                  55
7       mangue                 100
8       abricot                11
9       citron                 99


NbAssociations =11

Indice  Cle                   Valeur

1       orange                 88
2       prune                  55
3       ananas                 33
4       mangue                 100
5       poire                  77
6       citron                 99
7       amande                 22
8       papaye                 11
12      pomme                  44
13      griotte                66
```

```
14      abricot                 11

Comptage des mots dans un texte
le chat poursuit la souris ah quel chat ah quel texte ah ah
Indice  Cle                     Valeur

0       le                      1
1       poursuit                1
2       chat                    2
3       la                      1
4       quel                    2
5       ah                      4
7       texte                   1
9       souris                  1

*/
```

Essai.cc

```
#include "Dico.cc"
#include <string>

//def de hash : a placer avant l'instanciation  de la classe!
int Dico<string,int>::hash(string C, int tailleTab)
{int i=0;
 for (int j=0; j<C.length(); j++)i+=(j+1)*C[j];
 return (i % tailleTab);
 }

string Dico< string,int > ::cleDefaut="rien";

int Dico< string,int > ::valeurDefaut=0;

int valeurDefaut = 0;

template class Dico<string, int>;

template ostream& operator<<(ostream& , const Dico<string, int>& );

/*
ce fichier suppose que Dico.h contient les declarations:

  static TypeCle cleDefaut; //valeur par defaut pour la cle
  static TypeValeur valeurDefaut; //valeur par defaut pour la valeur

  static int hash(TypeCle, int); //methode de hachage

ces deux variables et cette méthode sont necessaires pour ecrire le code
des diverses methodes dans Dico.cc, mais on ne peut fixer les valeurs
de ces variables et le contenu de cette methode que lorsqu'on connait
les types effectifs de cle et valeur.
On n'a donc rien dans Dico.cc qui correspond a leurs declarations dans
Dico.h, mais on doit le faire dans DicoStringInt.cc avant d'ecrire la ligne
qui cree vraiment la classe:
  template class Dico<string,int>;
*/


/*
sur les 2 lignes d'init des attributs de classe:
essai.cc:11: erreur: trop peu de patron de listes de paramètres
essai.cc:11: erreur: expected `,' or `;' avant un élément lexical « = »
essai.cc:13: erreur: trop peu de patron de listes de paramètres
essai.cc:13: erreur: expected `,' or `;' avant un élément lexical « = »
*/
```

etudMain.cc

```
using namespace std;
#include<iostream>
#include<string>
#include"Dico.h"

int main()
{

  Dico<string,int> D;  //par defaut, capacite  10, taille 0
  cout << "\nNbAssociations =" << D.taille() << endl;
  D.affiche(cout);cout <<endl;

  //
  D.put("abricot", 11);
  D.put("amande", 22);
  D.put("ananas",33);
  D.put("pomme", 44);
  cout << "\nNbAssociations =" << D.taille() <<endl;
  D.affiche(cout);cout <<endl;

  D.put("prune",55);
  D.put("griotte", 66);
  D.put("poire", 77);
  D.put("orange",88);
  D.put("citron",99);
  D.put("mangue",100);
  cout << "\nNbAssociations =" << D.taille() <<endl;
  D.affiche(cout);cout <<endl;
  // la le dico est plein

  //on ajoute une assoc: le dico allonge de 5=> taille 15
  D.put("papaye",11);
  cout << "\nNbAssociations =" << D.taille() <<endl;
  D.affiche(cout);cout <<endl;
}
/*

NbAssociations =0
Dico vide


NbAssociations =4

Indice  Cle                   Valeur

2       amande                  22
3       ananas                  33
4       pomme                   44
8       abricot                 11


NbAssociations =10

Indice  Cle                   Valeur

0       poire                   77
1       orange                  88
2       amande                  22
```

```
3       ananas                  33
4       pomme                   44
5       griotte                 66
6       prune                   55
7       mangue                  100
8       abricot                 11
9       citron                  99


NbAssociations =11

Indice  Cle                     Valeur

1       orange                  88
2       prune                   55
3       ananas                  33
4       mangue                  100
5       poire                   77
6       citron                  99
7       amande                  22
8       papaye                  11
12      pomme                   44
13      griotte                 66
14      abricot                 11

*/
```

Complements

dans ce repertoire:

```
Assoc.h et Assoc.cc: la classe generique Association  du cours

AssocStringInt.cc: le fichier intermediaire a utiliser pour la compilation
                   separee de la classe Association<string,int>

DicoStringInt.cc: un exemple de fichier intermediaire possible pour
                  la compilation separee de la classe
Dictionnaire<string.int>
                  a adapter a vos notations!

dicmain.cc: un petit main pour tester le minimum; il contient en
commentaires
            les resultats qu'il doit produire (a la presentation pres).

deroulement de la fabrication de l'executable:

g++ -c AssocStringInt.cc     produit AssocStringInt.o


apres avoir ecrit Dico.h et Dico.cc,
g++ -c DicoStringInt.cc     produit DicoStringInt.o


g++ -c dicmain.cc produit dicmain.o

g++ dicmain.o AssocStringInt.o DicoStringInt.o -o dicmain

produit un executable qui doit marcher si votre programme est correct...
```

dans ce repertoire:

```
les fichiers pour une version du Dico qui contourne la bug de la version
3.4.4 de g++...

Assoc.h et Assoc.cc: la classe generique Association  du cours

AssocStringInt.cc: le fichier intermediaire a utiliser pour la compilation
                   separee de la classe Association<string,int>

Dico2.h : les attributs cleDefaut et valeurDefaut sont devenus des
          attributs d'instance

DicoStringInt2.cc: un exemple de fichier intermediaire possible pour
    la compilation separee de la classe Dico<string.int> ,a adapter a vos
notations!
    Ce fichier contient le code du constructeur sans paramètres, défini
pour
    une cle de type string et une valeur de type int.
    Dans le Dico2.cc qui vous reste à écrire, ce constructeur doit etre
présent,
    mais avec un corps vide, puisque celui qui servira est celui du fichier
    DicoStringInt2.cc

dicmain.cc: un petit main pour tester le minimum; il contient en
commentaires
```

les resultats qu'il doit produire (a la presentation pres).

deroulement de la fabrication de l'executable:

g++ -c AssocStringInt.cc produit AssocStringInt.o

g++ -c DicoStringInt2.cc produit DicoStringInt2.o

g++ -c dicmain.cc produit dicmain.o

g++ dicmain.o AssocStringInt.o DicoStringInt2.o -o dicmain

produit un executable qui doit marcher si votre programme est correct...

Templates, scoping, and digraphs.

If you have a class in the global namespace, say named X, and want to give it as a template argument to some other class, say std::vector, then std::vector<::X> fails with a parser error.

The reason is that the standard mandates that the sequence <: is treated as if it were the token [. (There are several such combinations of characters - they are called digraphs.) Depending on the version, the compiler then reports a parse error before the character : (the colon before X) or a missing closing bracket].

The simplest way to avoid this is to write std::vector< ::X>, i.e. place a space between the opening angle bracket and the scope operator.

---

prov.txt: coucou, pour faire un fichier ALIRE dans /commun/info/iup2/C++

dico generique, version avec hash en methode statique
classes Assoc et Dico

etu.doq fichier d'explications pour le Tp, fourni avec les fichiers d'Assoc et DicoStringInt.cc; etumain.cc main pour etudiants (a appeler dicmain.cc
dans le transfert)

etucorrig.doq fichier d'explications pour le corrige;
dicmaincorrig.cc main complet.

---

etudCorrig.doc

vous avez deja vu dans ce repertoire
Assoc.h, Assoc.cc (classes du cours)

```
AssocStringInt.cc et DicoStringInt.cc
dicmain.cc exemple de main


vous trouverez aussi maintenant:
Dico.h et Dico.cc pour la classe Dico
dicmaincorrig.cc exemple de programme qui contient la fonction de comptage
des mots
(qui n'etait pas, evidemment dans dicmain.cc):
il n'est pas tres beau, il n'y a pas de fichier
 de donnees, tout est "en dur" dans le code; mais ca permet de voir que
 ca marche.
```