Travaux dirigés Informatique C++

Héritage des classes (TD 9)

On désire réaliser la conversion **inverse** de celle proposée par la classe **CThermocouple** :

température (**T** en °*C*)→ différence de potentiel ou ddp (**E** en *mV*)

par le calcul d'un polynôme : $E = \sum_{i=0}^{n} d_i T^i$

avec les 9 coefficients d_i suivants (par ordre de puissance croissante de 0 à 8) :

```
0.000000E+00
0.503811878150E-01
0.304758369300E-04
-0.856810657200E-07
0.132281952950E-09
-0.170529583370E-12
0.209480906970E-15
-0.125383953360E-18
0.156317256970E-22
```

Pour cela, on propose de créer une nouvelle classe **CIThermocouple**. On peut remarquer que cette classe possèdera de nombreuses fonctions communes avec la classe **CThermocouple** précédente:
- les mêmes données membres (nombre **NC** et table des coefficients **coef**);
- les mêmes constructeurs,
- le même destructeur,
- les mêmes fonctions membres pour lire ou sauvegarder les coefficients dans un fichier, ou encore les afficher à l'écran.

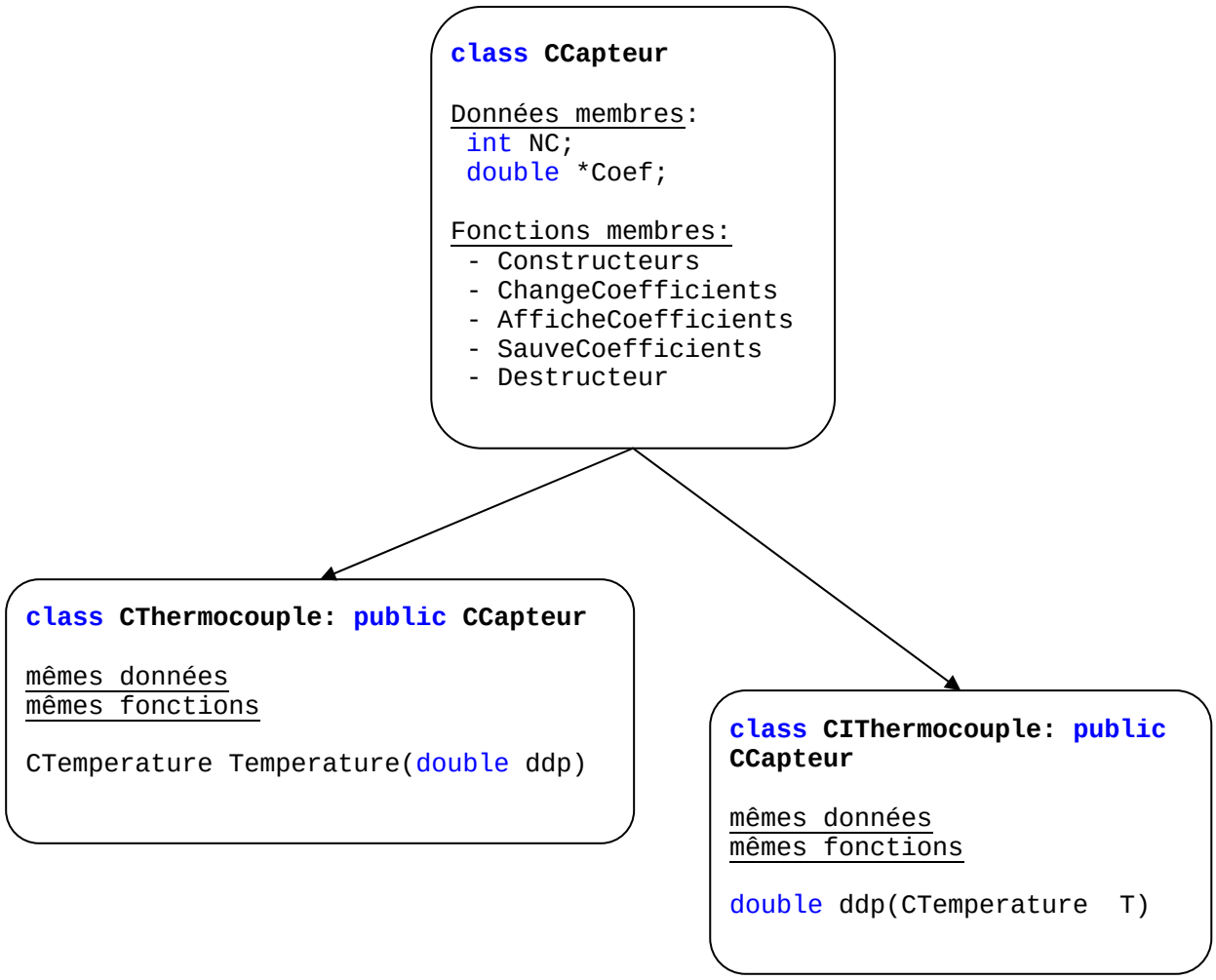
En fait, la seule différence réside dans la fonction de calcul du polynôme. Pour lequel, il faudra deux fonctions membres différentes :

```
CTemperature Temperature(double ddp);
```
pour **CThermocouple**

```
double ddp(CTemperature T);
```
pour **CIThermocouple**

Il est donc intéressant d'utiliser la notion d'héritage entre les classes, en créant une classe ancêtre **CCapteur**, qui regroupe les données et fonctions communes et faire « **hériter** » de ces propriétés les deux classes filles **CThermocouple** et **CIThermocouple**.

1) Architecture des classes



2) Code

2.1) Créer `class CCapteur` :

- click droit sur le projet → Ajouter → une classe C++
Nom de la classe **: CCapteur**

Vérifier dans l'explorateur de solution que l'assistant a bien ajouté les deux fichiers **Capteur.h** et **Capteur.cpp**

– dans **Capteur.h**, copier tout le code pris dans **CThermocouple.h**, modifier l'accès aux données membres en **protected** (pour permettre l'accès aux classes héritières). Puis supprimer la déclaration de la fonction **CTemperature Temperature(double ddp);**, et changer tous les noms des constructeurs et destructeurs

```
#pragma once
#include <iostream>
#include <fstream>
```

```
using namespace std;
#include "CTemperature.h"

// Classe ancêtre commune à CThermocouple et CIThermocouple

class CCapteur
{
protected:              // permet d'utiliser les données dans les classes filles
      int NC;
      double *Coef;
public:
      CCapteur(int n);
      CCapteur(char *nomfichier);
      bool ChangeCoefficients(int n);
      void ChangeCoefficients (char *nomfichier);
      bool AfficheCoefficients();
      void SauveCoefficients ();
      ~CCapteur(void);
};
```

Remarque : Sans précision, les données membres sont par défaut **private** donc seulement accessibles par les fonctions membres de la classe. Les données membres déclarées comme **protected** sont accessibles par les fonctions membres de la classe mais aussi des classes héritières.

- dans **Capteur.cpp**, copier tout le code pris dans **CThermocouple.cpp**, et supprimer le code de la fonction `CTemperature Temperature(double ddp);`, et changer tous les noms dans les fonctions

```
#include "Capteur.h"

// constructeur par saisie clavier
CCapteur::CCapteur(int n)
{
Coef=0;
ChangeCoefficients(n);
}
// constructeur par lecture dans un fichier
CCapteur::CCapteur(char *nomfichier)
{
Coef=0;
ChangeCoefficients(nomfichier);
}

// fonction pour saisir les coefficients au clavier
bool CCapteur::ChangeCoefficients(int n)
{
      if (Coef) delete [] Coef;
      NC=n;
      Coef = new double [NC]; // on réserve (alloue) la place mémoire nécessaire
      if (Coef)
            {
            for (int i=0;i<NC;i++) {
             cout<<endl<<"Coef["<< i << "]: ";
         cin >> Coef[i];
            }
            return true;
            }
```

```
      else return false;
}

// fonction pour lire les coefficients dans un fichier
void CCapteur::ChangeCoefficients (char *nomfichier)
{
ifstream in(nomfichier);
if (in.good())                // suppose que le fichier existe
      {if (Coef) delete [] Coef;
       in >> NC;
       Coef = new double[NC];
       if (Coef) for (int i=0; i<NC; i++) in >> Coef[i];
       in.close();
      }
else cout <<endl<<"impossible d'ouvrir le fichier "<<nomfichier<<endl;
}

//  fonction d'affichage des coefficients
bool CCapteur::AfficheCoefficients()
{
      if (Coef)
            {
            for (int i=0;i<NC;i++) cout << endl << "Coef" << i << ": " << Coef[i];
            cout<<endl;
            return true;
            }
      else return false;
}

// fonction pour sauvegarder les coefficients dans un fichier
void CCapteur::SauveCoefficients ()
{
char *nomfichier;
cout<<endl<<"saisir le nom du fichier: ";
nomfichier = new char[50];
cin >> nomfichier;
ofstream f(nomfichier); // ouverture en écriture
if (f.good())                 // suppose que le fichier existe
      {f << NC <<endl;
       if (Coef) for (int i=0; i<NC; i++) f << Coef[i]<<endl;
       f.close();
      }
else cout<<endl<<"impossible d'ouvrir le fichier "<<nomfichier<<endl;
}

// destructeur du tableau dynamique
CCapteur::~CCapteur()
{
      if (Coef) delete [] Coef;
}
```

2.2) Faire hériter `class CThermocouple` de la `class CCapteur`

– dans **CThermocouple.h**, ajouter l'instruction :
`class CThermocouple: public CCapteur`,
puis supprimer les données membres et ne conserver que les constructeurs, destructeur et la fonction spéciale `CTemperature Temperature(double ddp);`

```
// Fichier CThermocouple.h

#include "Capteur.h"

class CThermocouple: public CCapteur            // hérite de la classe CCapteur
{
public : // on garde la déclaration des contructeurs et destructeur et fonction
spéciale
      CThermocouple(int n);
      CThermocouple(char *nomfichier);
      CTemperature Temperature(double ddp);     //seule fonction à conserver
      ~CThermocouple(); // destructeur nécessaire si allocation dynamique
};
```

- dans **CThermocouple.cpp**, faire hériter les constructeurs et supprimer le code qui est déjà décrit dans **Capteur.cpp**. Le destructeur est appelé automatiquement. On conserve la fonction spéciale.

```
// Fichier CThermocouple.cpp

#include "CThermocouple.h"

// constructeur par tableau dynamique : appel celui de CCapteur
CThermocouple::CThermocouple(int n):CCapteur(n)
{}
// constructeur par lecture dans fichier : appel celui de CCpateur
CThermocouple::CThermocouple(char *nomfichier):CCapteur(nomfichier)
{}

// par héritage on accède aux fonctions membres de CCapteur

// fonction spéciale
CTemperature CThermocouple::Temperature(double ddp)
{
      double T = Coef[NC-1];
      for (int i=NC-2;i>=0;i--) T = T*ddp + Coef[i];
      CTemperature C;
      C.fixeC(T);
      return C;
}
// destructeur du tableau dynamique
CThermocouple::~CThermocouple()                 // appel automatique
{}
```

2.3) Créer `class CIThermocouple`

- Click droit sur le projet → Ajouter → une classe C++
Nom de la classe : **CIThermocouple**

Vérifier dans l'explorateur de solution que l'assistant a bien ajouté les deux fichiers **IThermocouple.h** et **IThermocouple.cpp**

– dans **IThermocouple.h**, copier tout le code pris dans **CThermocouple.h**, modifier le nom de la classe partout et la déclaration de la fonction spéciale

```
// Fichier IThermocouple.h

#include "Capteur.h"

class CIThermocouple: public CCapteur          // hérite de la classe CCapteur
{
public : // on garde la déclaration des contructeurs et destructeur et fonction
spéciale
      CIThermocouple(int n);
      CIThermocouple(char *nomfichier);
      double ddp(CTemperature T);        //seule fonction à modifier
      ~CIThermocouple();     // destructeur nécessaire si allocation dynamique
};
```

– dans **IThermocouple.cpp**, copier tout le code pris dans **CThermocouple.cpp**, modifier le nom de la classe partout et remplacer le code de la fonction spéciale

```
// Fichier IThermocouple.cpp

#include "IThermocouple.h"

// constructeur par tableau dynamique : appel celui de CCapteur
CIThermocouple::CIThermocouple(int n):CCapteur(n)
{}
// constructeur par lecture dans fichier : appel celui de CCpateur
CIThermocouple::CIThermocouple(char *nomfichier):CCapteur(nomfichier)
{}

// par héritage on accède aux fonctions membres de CCapteur

// fonction spéciale
double CIThermocouple::ddp(CTemperature T)
{
      double temp = T.TempC();
      double ddp = Coef[NC-1];
      for (int i=NC-2;i>=0;i--) ddp = temp*ddp + Coef[i];
      return ddp;
}
// destructeur du tableau dynamique
CIThermocouple::~CIThermocouple()              // appel automatique
{}
```

Recompiler et tester cette solution.

Modifier le programme principal pour faire une double conversion

ddp → T et ***T → ddp***

avec ces deux classes.