TD 6 – Un examen

Préparation individuelle :

au mieux exercice (1h max !)

au pire questions 3, 4, 6, 8 et 9 (30 min max)

Le but de la préparation est d'auto évaluer votre capacité à répondre à l'exercice d'un examen dans le temps imparti.

Au travers de l'exercice, les points suivants seront abordés :

- le polymorphisme,
- la gestion d'objets dynamique (avec la classe *std ::vector*)
- un mécanisme de chainage proche des listes.

Le but de la séance est de réaliser sous QtCreator votre application (les sources à compléter sont données).

Les étudiants les plus avancés pourront utiliser les outils graphiques vus en 3GE (IF1). pour tracer les diagrammes de Bode.

Il est possible d'étendre cette modélisation aux quadripôles, puis d'ajouter des éléments actifs (sources tension ou courant). Dans ce cas, le calcul d'impédance sera complété par ceux des tensions et courants.

Enfin, les valeurs des éléments peuvent aussi dépendre de grandeurs mesurées dans le circuit…

Examen Méthodologie Orientée Objets

documents autorisés

Exercice : Calcul d'impédances R, L, C (50 minutes)

Le but est de concevoir une application capable de calculer, pour une fréquence donnée, l'impédance complexe d'un circuit. Un circuit ne sera composé que de résistances, capacités et inductances. Les composants pourront être associés en série et parallèle, ou en différents mélanges de ces constructions (pas de boucle, ni de potentiel).
La figure ci-dessous donne des exemples de circuits « calculables » avec cette application.

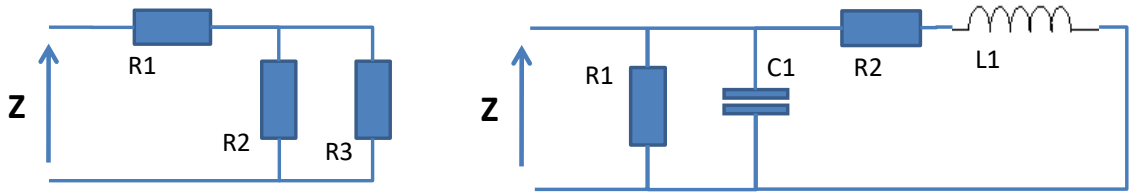


Figure 1: à gauche a) exemple avec 3 résistances; à droite, b) circuit RLC parallèle

Voici comment chaque composant sera représenté :

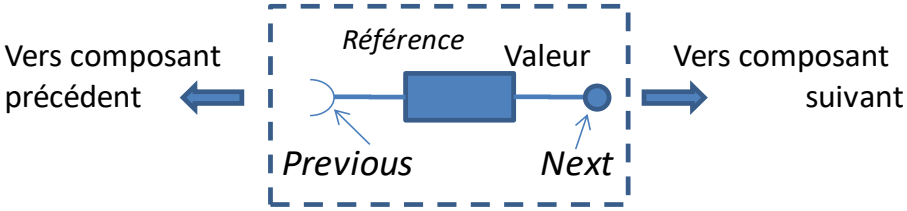


Figure 2: Schéma d'un composant

La création d'un circuit se fera via la mise bout-à-bout de plusieurs composants et par la création d'une « impedance ». C'est cette « impedance » qui permettra le calcul de l'impédance totale du circuit (à une fréquence donnée). Elle permettra également la création de branches parallèles (figure 3).

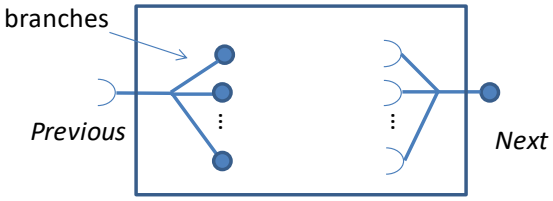
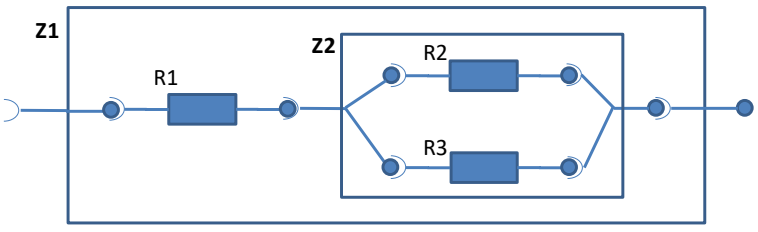


Figure 3: Schema d'"Impedance"

Comme exemple, voici la vue correspondant au premier circuit (fig1.a) et le code C++ permettant de le créer, l'afficher et le calculer pour la fréquence 0Hz.




```
Resistor R1("R1",50),
   R2("R2",100),
   R3("R3",100);
Impedance Z1("",0);
Impedance Z2("",0);
Z2.AddBranch(&R2);
Z2.AddBranch(&R3);
Z2.SetPrevious(&R1);
Z1.AddBranch(&R1);
Z1.Print();
Z1.Evaluate(0).Print();
```

Figure 4: Modélisation du circuit de la fig1.a et implémentation.

Partie 1 : Modélisation des composants R, L et C (15 minutes)
La modélisation des composants sera basée sur l'utilisation de la classe « ***Component*** » dont le code C++ est donné ci après.

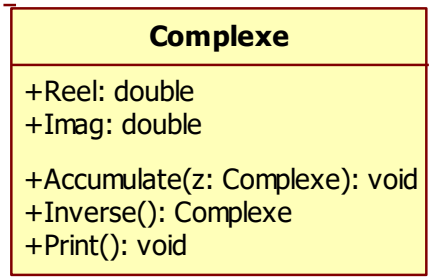
```
class Component
{
private:
      Component   *Previous;
      Component   *Next;
public:
      double      Valeur;
      string      Name;
      Component(string name, double v): Name( name ), Valeur( v )
            { Reset();  }
      void Reset()
            { Previous = NULL; Next = NULL;     }

      Component*  GetPrevious() { return Previous; }
      Component*  GetNext() { return Next; }

      void  SetPrevious( Component *comp )
            {// unlink
            if( Previous != NULL )
                  Previous->Next=comp;
            // make new link
            Previous = comp;
            Previous->Next = this;
            }
      virtual Complexe  Evaluate(double freq)=0;
      virtual void Print() { cout <<Name<<"("<<Valeur<<")"; }
};
```

1) Donner le diagramme UML de cette classe.
2) Quel type de relation lie *Previous* et *Next* à la classe « ***Component*** » ?
3) Peut-on créer un objet de type « Component » ? Quelle est la spécificité de cette classe ?
4) En vous appuyant sur la classe ***Component***, modéliser les classes ***Resistor***, ***Capacitor*** et ***Inductor*****. Justifier vos choix (méthodes, champs, associations).**
5) Ecrire le code C++ de la fonction *Evaluate* de la classe ***Capacitor***. Cette fonction simple retourne l'impédance ***Complexe*** du composant courant calculée à la fréquence passée en paramètre. La classe ***Complexe*** à utiliser est donnée ci-après.

Partie 2 : Classe Complexe (10 minutes)
On étudie la classe ***Complexe*** utilisée pour les calculs de cette application.




```
class Complexe
{
public:
      double Reel;
      double Imag;
      void Accumulate( Complexe z )
            { Reel += z.Reel;
              Imag += z.Imag; }
      Complexe Inverse() ; // = 1/z
      void Print() ;
};
```

6) En utilisant des objets de cette classe, donner le code C++ du calcul de l'impédance (complexe) totale de 3 impédances (complexes) en parallèles.
7) Modifier l'exemple de la question 6 en utilisant les fonctions *Evaluate* d'un objet ***Resistor***, un objet ***Capacitor*** et un objet ***Inductor*** ainsi qu'une fréquence (choisir les valeurs arbitrairement).

Partie 3 : Classe « Impedance » (15 minutes)

On s'intéresse maintenant à la classe ***Impedance***, qui est la classe centrale dans le calcul d'impédance des circuits série et parallèle. Cette classe utilise la classe **vector** pour la gestion dynamique des branches. On donne comme extrait de cette classe la fonction *Print()* qui affiche le nom et valeur de tous les composants contenu dans l'impédance :

```
void Impedance ::Print()
            {
            cout<<Name<<"[ ";
            if( !Branch.empty() ) // il y a des branches ?
            for(unsigned int i=0; i<Branch.size(); i++)
                  {
                  Component *comp = Branch[i];
                  do    {
                        comp->Print();
                        comp = comp->GetNext();
                        if( comp != NULL ) cout<<" + ";
                        }while( comp != NULL ); // branch pas finie ?
                  if( (Branch.size() > 1) && (i < Branch.size()-1))
                         cout<< " // ";
                  }
            cout<<"]";
            }
```

8) Justifier le fait que la classe ***Impedance*** hérite de ***Component***.
9) Donner le diagramme UML de la classe ***Impedance***. (cf. introduction !)
10) Ecrire en C++ la fonction *Evaluate()* de la classe ***Impedance***. (cf. questions 6 et 7)
*Pour le TD la classe **Impedance** est à faire.*

Fin.