Université de Nice-Sophia Antipolis — POLYTECH
ELSE4-FISE — 2024–2025

Examen de C++

Durée : 1h00
Aucun document autorisé
Mobiles interdits

▶ 1. À quoi correspond la notion d'*héritage* dans un langage OO comme C++ ?

---

voir cours

.....

▶ 2. Expliquez la notion de *liaison dynamique* des langages OO.

---

voir cours

.....

On souhaite représenter des dipôles électriques élémentaires tels que les *résistances*, *bobines* et *condensateurs*. La valeur r d'une résistance est exprimée en ohms Ω. L'inductance L d'une bobine est exprimée en henrys H. Enfin, la capacité c d'un condensateur est en farad F.

L'*impédance* électrique mesure l'opposition du dipôle au passage d'un courant alternatif sinusoïdal. Pour un courant de pulsation ω, c'est un nombre *complexe*, noté z, tel que pour une résistance, $z = r$, pour une bobine, $z = i(\omega * l)$ et pour une condensateur, $z = i(-1/(\omega * c))$. On souhaite calculer l'impédance de ces dipôles.

▶ 3. Écrivez les classes `dipôle`, `résistance`, `bobine` et `condensateur`, munies de la méthode `getImpédance` qui prend en paramètre la pulsation ω de type réel **double**. Réfléchissez bien à l'organisation des classes. Vous utiliserez la classe `complexe` vue en TD (voir annexe).

```
/*
 *  classe abstraite pour représenter des dipôles unitaires
 *  composés. La méthode getImpédance est à redéfinir dans les
 *  classes héritières pour calculer l'impédance du dipôle courant
 */
class dipôle {
public:
  virtual ~dipôle() {}
  virtual complexe getImpédance(const double oméga) const=0;
};

class résistance : public dipôle {
  double r; // en ohms
public:
  // constructeur
  résistance(const double _r) : r(_r) { assert(_r>=0); }
  //
  virtual complexe getImpédance(const double oméga) const override {
    return complexe(this->r);
  }
};

class bobine : public dipôle {
  double l; // inductance en henrys
public:
  // constructeur
  bobine(const double _l) : l(_l) { { assert(_l>=0); }
  //
  virtual complexe getImpédance(const double oméga) const override {
    return complexe(0.0, oméga*this->l);
  }
};

class condensateur : public dipôle {
  double c; // capacité en farad
public:
  // constructeur
  condensateur(const double _c) : c(_c) { assert(_c>=0); }
  //
  virtual complexe getImpédance(const double oméga) const override {
    assert(oméga!=0.0);
    return complexe(0.0, -1/(oméga*this->c));
  }
};
```


.....

Les dipôles peuvent être *montés* en *série* ou en *parallèle*. En série, l'impédance z du montage est la somme des impédances des dipôles. Pour n dipôles en parallèle, l'impédance z est :

$$z = \frac{1}{\sum_{i=1}^{n} \frac{1}{z_i}}$$

▶ 4. Par héritage, écrivez les classes `dipôleComposéEnSérie` et `dipôleComposéEnParallèle` qui implémentent la méthode `getImpédance` pour calculer l'impédance d'un montage en série ou en parallèle. Les dipôles des montages seront mémorisés dans un `std::vector<dipôle *>`. Vous écrirez la méthode `ajouterDipôle` qui ajoute un `dipôle*` dans le vecteur. *Note* : vous pourrez définir une classe auxiliaire si vous le souhaitez (mais conseillé).

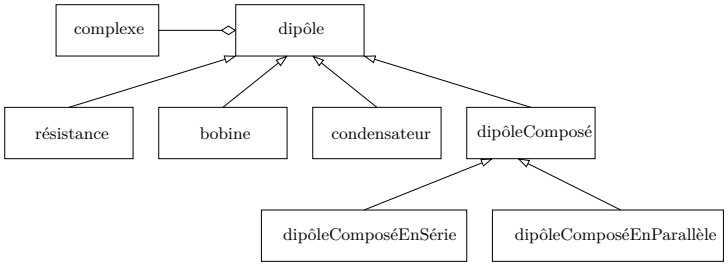
```
/*
 * classe mère des classes de composition de dipôle en série ou en parallèle
 * pour factoriser et n'avoir qu'une seule déclaration de vecteur et
 * de méthode d'ajout des dipôles
 */
class dipôleComposé : public dipôle {
protected:
  std::vector<dipôle*> lesDipôles;
public:
  void ajouterDipôle(dipôle *d) { this->lesDipôles.push_back(d); }
};
```

```
class dipôleComposéEnSérie : public dipôleComposé {
public:
  virtual complexe getImpédance(const double oméga) const override {
    complexe impédance;
    for (const auto x : dipôleComposé::lesDipôles)
      impédance = impédance + x->getImpédance(oméga);
    //
    return impédance;
  }
};

class dipôleComposéEnParallèle : public dipôleComposé {
public:
  virtual complexe getImpédance(const double oméga) const override {
    complexe impédance;
    for (const auto x : dipôleComposé::lesDipôles)
      impédance += x->getImpédance(oméga).inverse();
    //
    return impédance.inverse();
  }
};
```

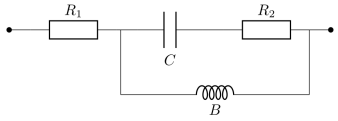
.....

► 5. Dessinez le diagramme des classes que vous avez écrites.



.....

► 6. Écrivez la méthode `main` qui construit le montage donné ci-dessous avec $R1 = 100\,\Omega$, $R2 = 10\,\Omega$, $C = 10^{-6}\,F$, $B = 10^{-2}\,H$; puis qui écrit sur la sortie standard, toutes ses impédances pour des pulsations ω comprises entre 1 et $pulsationMax = 20\ 000\,rd/s$. Pour chaque pulsation, vous écrirez l'impédance en coordonnées polaires (*i.e.* le module suivi de l'argument).



```
int main() {
  const double pulsation = 10000;
  résistance R1(100), R2(10); // R1 = 100 ohms, R2 = 10 ohms,
  condensateur C(1e-6);       // C = 1 microfarad
  bobine B(1e-2);             // B = 10 mH

  // mettre en série le condensateur B et la résistance R2
  dipôleComposéEnSérie série1;
  série1.ajouterDipôle(&C); série1.ajouterDipôle(&R2);
  // mettre en // série1 et la bobine B
  dipôleComposéEnParallèle parallèle;
  parallèle.ajouterDipôle(&série1);  parallèle.ajouterDipôle(&B);
  // mettre en série la résistance R1 et le montage en //
  dipôleComposéEnSérie série2;
  série2.ajouterDipôle(&R1); série2.ajouterDipôle(&parallèle);
  // afficher les impédances du montage entre
  // 1 et pulsationMax en coordonnées polaires
  for (int i=1; i<=pulsationMax; i++) {
    complexe z = série2.getImpédance(i);
    std::cout <<  i << " " << z.rho() << " " << z.theta() << std::endl;
  }
  //
  return EXIT_SUCCESS;
}
```

.....