Exo1 :

Q1 :

```
#include <iostream>
#include <cmath>

class Complex {
private:
    double part_reelle;
    double part_imaginaire;


public:
```

Q2 :

```
public:
    // Constructeurs
    Complex() : part_reelle(0.0), part_imaginaire(0.0) {} // Constructeur par défaut
    Complex(const Complex& other) : part_reelle(other.part_reelle),
part_imaginaire(other.part_imaginaire) {} // Constructeur par copie
```

Q3 :

```
 // Accesseurs et mutateurs
    double get_re() const {
        return part_reelle;
    }
    double get_im() const {
        return part_imaginaire;
    }

    void set_re(double reelle) {
        part_reelle = reelle;
    }

    void set_im(double imaginaire) {
        part_imaginaire = imaginaire;
    }
```

Q4 :

```
int main() {
    Complex c1; // Constructeur par défaut
    Complex c2(c1); // Constructeur par copie

    std::cout << "c1 : " << c1.get_re() << " + " << c1.get_im() << "i" << std::endl;
    std::cout << "c2 : " << c2.get_re() << " + " << c2.get_im() << "i" << std::endl;


    c2.set_re(2.5); // Modification de la partie réelle
```

```
    c2.set_im(-1.3); // Modification de la partie imaginaire

    std::cout << "c2 après la modif : " << c2.get_re() << " + " << c2.get_im() << "i" <<
std::endl;
```

Q5 :

```
 // Méthode pour obtenir le module
        double get_mod() const {
        return sqrt(part_reelle * part_reelle + part_imaginaire * part_imaginaire);
}
```

Q6 :

```
 // Méthode pour obtenir l'argument
    double get_arg() const {
        return atan2(part_imaginaire, part_reelle);}
```

Q7 :

```
Complex c3;
    c3.set_mod(5);
    c3.set_arg(3.14 / 4); // 45 degrees in radians
    std::cout << "c3 : " << c3.get_mod() << " + " << c3.get_arg() << "i" << std::endl;
```

Le test :

```
c1 : 0 + 0i
c2 : 0 + 0i
c2 après la modif : 2.5 + -1.3i
c3 : 5 + 0.785i
```

Q8 :

```
   void set_mod(double mod) {
        double arg = get_arg();
        part_reelle = mod * cos(arg);
        part_imaginaire = mod * sin(arg);}
```

Q9 :

```
 void set_arg(double arg) {
        double mod = get_mod();
        part_reelle = mod * cos(arg);
        part_imaginaire = mod * sin(arg);
    }
```

Q10 :

```
 void set_arg(double arg) {
        double mod = get_mod();
        part_reelle = mod * cos(arg);
        part_imaginaire = mod * sin(arg);}
```

Q11 :

```
#include <iostream>
#include <cmath>

class Complex {
private:
    double mod;
    double arg;

public:{
} ;
```

Q12 :

```
public:
    // Constructeurs
    Complex() : mod(0.0), arg(0.0) {} // Constructeur par défaut
    Complex(const Complex& other) : mod(other.mod), arg(other.arg) {} // Constructeur par
copie

    // Accesseurs et mutateurs
    double get_mod() const {
        return mod;
    }

    double get_arg() const {
        return arg;
    }

    void set_mod(double module) {
        mod = module;
    }

    void set_arg(double argument) {
        arg = argument;
    }
```

Q13

```
int main() {
    Complex c1; // Constructeur par défaut
    Complex c2(c1); // Constructeur par copie

    std::cout << "c1 : " << c1.get_mod() << " + " << c1.get_mod() << "i" << std::endl;
    std::cout << "c2 : " << c2.get_arg() << " + " << c2.get_arg() << "i" << std::endl;

    c2.set_mod(2.5); // Modification du module
    c2.set_arg(-1.3); // Modification de l'argument

    std::cout << "c2 after modification : " << c2.get_mod() << " + " << c2.get_arg() <<
"i" << std::endl;
```

Q14 :

```
 double get_re() const {
        return mod * cos(arg);
    }

    double get_im() const {
        return mod * sin(arg);
    }
} ;
```

Q15 :

```
 void set_re(double reelle) {
        double imaginaire = get_im();
        mod = sqrt(reelle * reelle + imaginaire * imaginaire);
        arg = atan2(imaginaire, reelle);
    }

    void set_im(double imaginaire) {
        double reelle = get_re();
        mod = sqrt(reelle * reelle + imaginaire * imaginaire);
        arg = atan2(imaginaire, reelle);
    }
```

Q16 :

Après avoir ajouté ces commandes on va compiler :

```
   // Affichage de la partie réelle et imaginaire
    std::cout << "Partie réelle de c1 : " << c1.get_re() << std::endl;
    std::cout << "Partie imaginaire de c1 : " << c1.get_im() << std::endl;

    // Modification de la partie réelle de c1
    c1.set_re(4);

    // Affichage de la nouvelle partie réelle et imaginaire de c1
    std::cout << "Après modification, partie réelle de c1 : " << c1.get_re() << std::endl;
    std::cout << "Après modification, partie imaginaire de c1 : " << c1.get_im() <<
std::endl;

    // Modification de la partie imaginaire de c1
    c1.set_im(-2);

    // Affichage de la nouvelle partie réelle et imaginaire de c1
    std::cout << "Après modification, partie réelle de c1 : " << c1.get_re() << std::endl;
    std::cout << "Après modification, partie imaginaire de c1 : " << c1.get_im() <<
std::endl;
```

le test :

c1 : 0 + 0i

c2 : 0 + 0i

c2 after modification : 2.5 + -1.3i

Partie réelle de c1 : 0

Partie imaginaire de c1 : 0

Après modification, partie réelle de c1 : 4

Après modification, partie imaginaire de c1 : 0

Après modification, partie réelle de c1 : 4

Après modification, partie imaginaire de c1 : -2

c3 : 5 + 0.785i