ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Département de Génie Informatique et Génie Logiciel

Cours INF1010 : Programmation orientée objet (Hiver 2015)

CONTRÔLE PÉRIODIQUE
CORRIGÉ

DATE : Le lundi 23 février 2015

HEURES : de 18h30 à 20h20

DUREE : 1h50

PONDERATION : 30%

LIEU:

Section 1: B-415 et A-410.
Section 2: B-418 et B 429

Documentation, calculatrice et appareils électroniques sont interdits.

Vous devez répondre sur le cahier.

Cet examen comprend 5 questions sur 14 pages pour un total de 20 points.

Question 1 – Questions en vrac (4 points)

Répondre aux questions suivantes en *une phrase au maximum*.

(a) **(½ point)** Lorsqu'un paramètre ou une variable possède le même nom qu'un attribut, est-il tout de même possible d'accéder à cet attribut dans une méthode ? Si oui, comment ?

Solution:
Oui, c'est possible. On utilise le pointeur `this` qui pointe vers l'objet courant et nous permet donc d'accéder à ses attributs. Si l'on prend par exemple un attribut `att_`, il sera possible d'y accéder via `this->att_`.

(b) **(½ point)** Dans le code suivant, considérant que la classe `Dragon` est fonctionnelle et possède tous les constructeurs nécessaires, combien de fois le constructeur par copie de la classe `Dragon` sera-t-il appelé ?

```
void fonction1(Dragon dragon1) {
    Dragon dragon2;
}

Dragon fonction2(const Dragon& dragon2) {
    return (*new Dragon());
}

int main() {
    Dragon dragon1;
    Dragon dragon2(dragon1);
    Dragon dragon3 = dragon1;

    Dragon dragon4;
    dragon3 = dragon2;

    fonction1(dragon1);
    dragon4 = fonction2(dragon1);
    fonction1(dragon4);

    return 0;
}
```

Solution:
Le constructeur par copie de la classe `Dragon` est appelé cinq (5) fois lors de l'exécution de ce programme.

(c) **(½ point)** Pour surcharger l'opérateur `<<`, on doit retourner une référence à l'objet dans le `return` de cette surcharge. De quelle classe de l'objet s'agit-il, et pourquoi devons nous retourner cette référence ?

Solution:
Il s'agit de l'objet de classe `ostream`. Cela permet d'afficher plusieurs objets (hétérogènes ou non) à la fois. C'est le principe d'appel en cascade.

(d) **(½ point)** Donnez la définition de la surcharge de l'opérateur `+=` de la classe `Complexe` prenant en paramètre un autre objet `Complexe`. Cette surcharge doit être une méthode de la classe.

Solution:

```
Complexe& operator+=(const Complexe &complexe);
```

(e) **(½ point)** Dans l'implémation de l'opérateur `=`, Comment vérifier que l'on n'affecte pas un objet à lui-même ?

Solution:
On utilise le pointeur `this`, que l'on compare à la référence de l'objet reçu en paramètre.

(f) **(½ point)** Dans le code suivant, considérant que la classe `Poney` est fonctionnelle et possède tous les constructeurs nécessaires, combien de blocs mémoires contenant un objet de classe `Poney` ont été alloués ?

```
int main() {
    Poney poney1("Granny Smith");

    Poney groupePoney[6];
    groupePoney[1] = Poney();

    Poney poney2;
    groupePoney[0] = poney1;
    groupePoney[2] = poney2;

    Poney poney3 = Poney();
    Poney* poney4 = new Poney();

    groupePoney[3] = poney3;
    groupePoney[4] = *poney4;

    return 0;
}
```

Solution:
Onze (11) blocs mémoire contenant un `Poney` ont été alloués.

(g) **(½ point)** Soient deux classes A et B, si l'on vous demande de choisir conceptuellement entre une relation d'héritage ou une relation de composition entre A et B, quelle est votre stratégie sur le choix de relation ?

Solution:
Si on choisi une relation de composition entre A et B, cela signifie que A « a un » B (ou inversement). Si on choisi une relation d'héritage entre A et B, cela signifie que A « est un » B (ou inversement).

(h) **(½ point)** Quel sera l'affichage du programme suivant ?

```
int main()
{
    vector<int> vect(9);

    vect.push_back(3);
    vect[5] = 8;
    vect.push_back(vect[1]);

    for (int i = 0; i < 3; i++) {
        vect.pop_back();
    }
    vect[2] = 6;
    vect.push_back(vect.back());
```

```
    vect.push_back(vect[1]);
    vect[9] = 7;

    cout << vect.size() << endl;

    return 0;
}
```

Solution:
Le programme affichera : `10`

Question 2 – Concepts de base (5 points)

L'un de vos amis propriétaire d'un cirque vous demande de l'aide. Il a tenté de modéliser son cirque en C++ et fait appel à vos connaissances acquises en INF1010 pour savoir s'il a correctement écrit son code (c'est à dire qu'il faut respecter tous les concepts du C++). Soient les classes et la fonction `main` suivantes :

```
class Employe {
public:
    string nom;
}

class Clown : public Employe {
public:
    string nom;
    string couleurDuNez;
}

class Animal {
public:
    string race;
}

class Cirque {
public:
    vector<Clown> clowns;
    vector<Animal*> animaux;
    Employe& directeur;
```

```
    void spectacleDeCeSoir() {
        cout << "Les clowns sont:" << endl;
        for (int i = 0; i < clowns.size(); i++) {
            cout << " - " << clowns[i].nom << " dont le nez est " <<
                clowns[i].couleurDuNez << endl;
        }
        cout << "Et les animaux sont:" << endl;
        for (int i = 0; i < animaux.size(); i++) {
            cout << " - " << animaux[i].race << endl;
        }
        cout << "Sous la direction de " << directeur.nom << endl;
    }

    ~Cirque() {
        for (int i = 0; i < clowns.size(); i++) {
            delete clowns[i];
        }
        for (int i = 0; i < animaux.size(); i++) {
            delete animaux[i];
        }
    };
}

int main () {
    Cirque cirque;

    Employe directeur;
    directeur.nom = "Marcel";

    Animal* cheval = new Animal();
    cheval->race = "cheval";
    cirque.animaux.push_back(cheval);

    Clown* bozzo = new Clown();
    clown->nom = "Bozzo";
    clown->couleurDuNez = "rouge";
    cirque.clowns.push_back(bozzo);

    cirque.spectacleDeCeSoir();

    return 0;
}
```

Avec les concepts vus en cours et votre connaissance du Guide du codage, aidez votre ami en écrivant une version corrigée(définition et implémentation) de...

(a) **(½ point)** ... la classe `Employe`.

Solution:

```
class Employe {
public:
    Employe() { nom_ = ""; };
    void setNom(string nom) { nom_ = nom; };
    string getNom() const { return nom_ };
private:
    string nom_;
}
```

(b) **(1 point)** ... la classe `Clown`.

Solution:

```
class Clown : public Employe {
public:
    Clown() { couleurDuNez_ = "aucune"; };
    void setCouleurDuNez(string couleur) { couleurDuNez_ =
       couleurDuNez; };
    string getCouleurDuNez() const { return couleurDuNez_; };
private:
    string couleurDuNez_;
}
```

(c) **(½ point)** ... la classe `Animal`.

Solution:

```
class Animal {
public:
    Animal() { race_ = "Poulpe"; };
    void setRace(string race) { race_ = race; };
    string getRace() const { return race_; };
private:
    string race_;
}
```

(d) **(2 points)** ... la classe `Cirque` (utiliser l'aggrégation).

Solution:

```
class Cirque {
public:
    Cirque(Employe& directeur) {
        directeur_ = directeur;
    };

    void addClown(Clown* clown) {
        clowns_.push_back(clown);
    };
    // Acceptable:
    //void setClowns(vector<Clown*> clown) {
    //  clown_ = clown;
    //}
```

```
    vector<Clown*> getClowns() const {
        return clowns_;
    };

    void addAnimal(Animal* animal) {
        animaux_.push_back(animal);
    };
    // Acceptable:
    //void setAnimaux(vector<Animal*> animaux) {
    //  animaux_ = animaux;
    //}

    vector<Animal*> getAnimaux() const {
        return animaux_;
    };

    void spectacleDeCeSoir() {
        cout << "Les clowns sont:" << endl;
        for (int i = 0; i < clowns_.size(); i++) {
            cout << " - " << clowns_[i]->getNom() << " dont le nez
                est " << clowns_[i]->getCouleurDuNez() << endl;
        }
        cout << "Et les animaux sont:" << endl;
        for (int i = 0; i < animaux_.size(); i++) {
            cout << " - " << animaux_[i]->getRace() << endl;
        }
        cout << "Sous la direction de " << directeur.getNom() <<
           endl;
    };

private:
    vector<Clown*> clowns_;
    vector<Animal*> animaux_;
    Employe& directeur_;
}
```

(e) **(1 point)** ... la fonction `main`.

Solution:

```
int main () {
    Employe directeur;
    directeur.setNom("Marcel");

    Cirque cirque(directeur);

    Animal* cheval = new Animal();
    cheval->setRace("cheval");
    cirque.addAnimal(cheval);
```

```
    Clown* bozzo = new Clown();
    clown->setNom("Bozzo");
    clown->setCouleurDuNez("rouge");
    cirque.addClown(cheval);

    cirque.spectacleDeCeSoir();

    return 0;
}
```

Question 3 – Composition et agrégation (3½ points)

Soit la classe `AA` :

```
class AA {
};
```

Pour chaque sous question, spécifier s'il s'agit d'une composition ou d'une agrégation.

(a) **(½ point)**

```
class BB {
public:
    BB(AA attrib) : attribB_(attrib){};
private:
    AA attribB_;
};

int main() {
    AA objetA;
    BB objetB(objetA);
    return 0;
}
```

Solution:
Composition.

(b) **(¾ point)**

```
class BB {
public:
    BB(AA * attrib) : attribB_(attrib){};
private:
    AA * attribB_;
};

int main() {
    AA * objetA = new AA();
    BB objetB(objetA);
    delete objetA;
    return 0;
}
```

Solution:
Aggrégation.

(c) **(¾ point)**

```
class BB {
public:
    BB(AA  attrib) : attribB_(new AA(attrib)){};
private:
    AA * attribB_;
};

int main() {
    AA objetA ;
    BB objetB(objetA);
    return 0;
}
```

Solution:
Composition.

(d) **(¾ point)**

```
class BB {
public:
    BB(AA & attrib) : attribB_(attrib){};
private:
    AA & attribB_;
};

int main() {
    AA objetA ;
    BB objetB(objetA);
    return 0;
}
```

Solution:
Aggrégation.

(e) **(¾ point)**

```
class BB {
public:
    BB(AA * attrib) : attribB_(attrib){};
private:
    AA * attribB_;
};

int main() {
    AA objetA ;
    BB objetB(&objetA);
    return 0;
}
```

Solution:
Aggrégation.

Question 4 – Héritage et composition (3 points)

Si on exécute ce programme, quel sera l'affichage ? Choisissez parmi les solutions d'affichage.

```
class AA {
public:
    AA(){ cout << "constructeur AA" << endl; }
};

class BB {
public:
    BB(){ cout << "constructeur BB" << endl; }
};

class CC {
public:
    CC(){ cout << "constructeur CC" << endl; }
private:
    BB attribC_;
};

class EE: public AA {
public:
    EE(){ cout << "constructeur EE" << endl; }
};

class DD: public AA {
public:
    DD(){ cout << "constructeur DD" << endl; }
private:
    EE attribD_;
};

class FF: public CC {
public:
    FF(){ cout << "constructeur FF" << endl; }
private:
    DD attribF_;
};

int  main() {
    FF uneInstance;
    return 0;
}
```

(a) **(1 point)** Quelles seront les deux (2) premières lignes ?

A.
```
constructeur CC
constructeur BB
```

B.
```
constructeur FF
constructeur DD
```

C.
```
constructeur BB
constructeur CC
```

D.
```
constructeur AA
constructeur BB
```

Solution:
La solution est le choix C.

(b) **(1 point)** Les trois (3) lignes suivantes ?

A.
```
constructeur EE
constructeur AA
constructeur AA
```

B.
```
constructeur AA
constructeur AA
constructeur EE
```

C.
```
constructeur AA
constructeur EE
constructeur AA
```

D.
```
constructeur CC
constructeur AA
constructeur FF
```

E.
```
constructeur FF
constructeur AA
constructeur DD
```

F.
```
constructeur AA
constructeur AA
constructeur FF
```

Solution:
La solution est le choix B.

(c) **(1 point)** Enfin, les deux (2) dernières lignes ?

A.
```
constructeur DD
constructeur FF
```

B.
```
constructeur AA
constructeur EE
```

C.
```
constructeur CC
constructeur BB
```

D.
```
constructeur EE
constructeur FF
```

Solution:
La solution est le choix A.

Question 5 – Héritage (4½ points)

Le réseau informatique est composé d'un ensemble de noeuds ayant chacun un rôle spécifique pour le fonctionnement du réseau : routeurs, un pare-feu, ordinateurs, serveurs. Chaque noeud possède une adresse IP sous la forme xxx.yyy.zzz.ttt et un nom de domaine.

Un serveur est une classe dérivée de la classe Noeud à laquelle on ajoute les informations permettant de conserver le protocole de communication du serveur, ainsi qu'un numéro de port représentant le service offert par ce serveur.

Soient les classes `AdresseIP`, `Noeud` et `Serveur` :

```
class AdresseIP {
public:
    AdresseIP(string adresseReseau);
    string getAdresse();
    void setAdresse(string adresseReseau);
    bool verifier(string adresseReseau);
    void afficher();
private:
    string adresseReseau_;
};

class Noeud {
public:
    Noeud(string adresseIP, string  nomDomaine);
    bool verifier(string adresseReseau, string nomDomaine);
    void afficher();
private:
    string nomDomaine_;
    AdresseIP adresseIP_;
};

class Serveur: public Noeud {
public:
    Serveur(string adresseIP, string nomDomaine, string protocole, int
       port);
    bool verifier(string adresseIP, string nomDomaine, string protocole,
       int port);
    void afficher();
private:
    int port_;
    string protocole_;
};
```

(a) **(1½ point)** i. **(½ point)** Écrire seulement l'implémentation du constructeur par paramètre qui initialise l'attribut `adresseReseau_` de la classe `AdresseIP`.

Solution:

```
AdresseIP::AdresseIP(string adresseReseau) {
    adresseReseau_= adresseReseau;
}
```

ii. **(½ point)** Écrire seulement l'implémentation du constructeur par paramètres qui initialise les attributs `adresseIP_` et `nomDomaine_` de la classe `Noeud`.

Solution:

```
Noeud::Noeud(string adresseIP, string nomDomaine): adresseIP_(
   adresseIP) {
    nomDomaine_= nomDomaine;
}
```

iii. **(½ point)** Écrire seulement l'implémentation du constructeur par paramètres qui initialise les attributs `adresseIP_`, `nomDomaine_`, `port_`, et `protocole_` de la classe `Serveur`.

Solution:

```
Serveur::Serveur(string adresseIP, string nomDomaine, string
   protocole, int port):Noeud(adresseIP,nomDomaine) {
    port_ = port;
    protocole_= protocole;
}
```

(b) **(1½ point)** i. **(½ point)** Écrire l'implémentation de la fonction membre `afficher` qui affiche la valeur de l'attribut `adresseReseau_` de la classe `AdresseIP`.

Solution:

```
void AdresseIP::afficher() {
    cout << " adresse Reseau "<< adresseReseau_<< endl;
}
```

ii. **(½ point)** Écrire l'implémentation de la fonction membre `afficher` qui affiche les valeurs des attributs `adresseIP_` et `nomDomaine_` de la classe `Noeud`.

Solution:

```
void Noeud::afficher() {
    adresseIP_.afficher();
    cout << "Nom du Domaine "<< nomDomaine_<< endl;
}
```

iii. **(½ point)** Écrire l'implémentation de la fonction membre `afficher` qui affiche les valeurs des attributs `adresseIP_`, `nomDomaine_`, `port_`, et `protocole_` de la classe `Serveur`.

Solution:

```
void Serveur::afficher() {
    Noeud::afficher();
    cout << "port " << port_ << "\t" << "protocole " <<
       protocole_ << endl;
}
```

(c) **(1½ point)** i. **(½ point)** Écrire l'implémentation de la fonction membre `verifier` qui compare la valeur de l'attribut `adresseReseau_` de la classe `AdresseIP` au paramètre de cette fonction membre. La fonction retourne le résultat de la comparaison.

Solution:

```
bool AdresseIP::verifier (string adresseReseau) {
    return adresseReseau_ == adresseReseau;
}
```

ii. **(½ point)** Écrire l'implémentation de la fonction membre `verifier` qui compare les valeurs des attributs `adresseIP_` et `nomDomaine_` de la classe `Noeud` aux paramètres de cette fonction membre. La fonction retourne le résultat de la comparaison.

Solution:

```
bool Noeud:: verifier (string adresseIP, string nomDomaine) {
    return adresseIP_.verifier(adresseIP) && (nomDomaine==
       nomDomaine_);
}
```

iii. **(½ point)** Écrire l'implémentation de la fonction membre `verifier` qui compare les valeurs des attributs `adresseIP_`, `nomDomaine_`, `port_`, et `protocole__` de la classe `Serveur` aux paramètres de cette fonction membre. La fonction retourne le résultat de la comparaison.

Solution:

```
bool Serveur::verifier(string adresseIP, string nomDomaine,
   string protocole, int port) {
    return Noeud::verifier(adresseIP, nomDomaine) &&
       protocole_== protocole && port_ == port;
}
```