ECE
PARIS ÉCOLE D'INGÉNIEURS

ECE 2015-2016
Conception et Programmation objet C++
ING 2 - Partiel

Merci de respecter les cadres pour les réponses. Tout débordement ne sera pas pris en compte et pourra être pénalisé de 1 point maximum par exercice.

Question de cours (5 points) / 1pt par question

1) Quelle est la différence entre une classe et un objet ?

Objet est une instance d'une classe
Ou alors : classe est un type et objet est celui qui utilise la classe

2) Qu'est-ce qu'une interface ?

Interface = partie publique du fichier header

3) A quoi sert la surcharge d'opérateur ?

A effectuer des opérations entre des OBJETS (et non des scalaires ou des classes...)

4) J'ajoute dans un tableau les entiers 1,2,3,4 et 5. Quelle structure de donnée utiliser pour récupérer les nombres dans l'ordre inverse ? Justifiez en une phrase.

Une pile car LIFO (Last In First Out)

5) Donner 2 des 4 avantages d'une référence par rapport aux pointeurs.

Pas d'allocation et désallocation mémoire
Garantie que l'objet sera non nul
S'utilise comme l'objet lui même (pas de flèche comme pour le pointeur)
Pas de fuite mémoire

Exercice 1 : Algorithmie et C++ (3 points)

Considérons un ensemble de 4 éléments E1, E2, E3 et E4.
On souhaite générer "2 parmi 4" combinaisons sans tenir compte de l'ordre. On obtiendra les combinaisons suivantes :

{ (E1, E2) (E1, E3) (E1, E4) (E2, E3) (E2, E4) (E3, E4) } = 6 combinaisons

Puisque l'ordre n'a pas d'importance, on a (E1, E2) = (E2, E1).

Créer un sous programme nommé *generateCombi* en C++ permettant de générer les combinaisons ci-dessus. On récupérera toutes les combinaisons dans un vecteur en sortie de la fonction.

1) Donner le prototype de la fonction *generateCombi* **(1 point)**
N'accepter que le double vecteur comme solution (cf énoncé)
-0.5pt si manque le ; à la fin du prototype ou tout autre petite erreur

```
Std::vector<std::vector<Elem>> generateCombi(); // NB: Elem peut être un int ou autre type
```

2) Implémenter *generateCombi* en C++ à l'aide de boucles **(2 points)**
Mettre 1pt si il y a deux boucles for avec les indices "corrects"
-0.5pt si return est oublié

```
std::vector<std::vector<Elem>> generateCombi() // dans mon cas Elem = int
{
  unsigned int size = 2; // car 2 éléments par combinaison
  std::vector<std::vector<Elem>> res;
  std::vector<int> _tmp(size, 0); // équivalent du resize()
  for (unsigned int i=0; i<size-1; ++i) // accepter j<size (au lieu de size-1)
  {
    for (unsigned int j=(i+1); j<size; ++j) // accepter j=i (au lieu de i+1)
    {
      _tmp[0] = i;
      _tmp[1] = j;
      res.push_back(_tmp);
    }
  }

  return res;
}
```

Exercice 2 : Compréhension de code de base (3 points)

```
class Base
{
    public:
        Base() { std::cout <<"Contructeur Base"<<std::endl; }
        ~Base() { std::cout <<"Destructeur Base"<<std::endl; }
};
```

```
class A : public Base
{
    public:
        A() { std::cout <<"Contructeur A"<<std::endl; }
        ~A() { std::cout <<"Destructeur A"<<std::endl; }
};

class B
{
    private:
        A* m_pt;
    public:
        B() { std::cout <<"Contructeur B"<<std::endl; }
        ~B() { std::cout <<"Destructeur B"<<std::endl; }
        void test() { m_pt = NULL; }
};

int main(int argc, char** argv)
{
    A* a = new A();
    B b;
    b.test();
    delete a;
    return 0;
}
```

Ecrire la sortie du programme (ce qui sera affiché à l'écran lors de son exécution).

0.5pt par ligne correcte dans le BON ORDRE.

Contructeur Base
Contructeur A
Constructeur B
Destructeur A
Destructeur Base
Destructeur B

Exercice 3 - Compréhension de code avancé (4 points)

```
class Date
{
    private:
        int m_year;
        int m_month;
        int m_day;
    public:
        Date(int year, int month, int day) : m_year(year), m_month(month), m_day(day) {}
        ~Date() {}

        void display() { std::cout << m_year << "-" << m_month << "-" << m_day << std::endl; }
        void setMonth(int m) { m_month = m; }

        bool operator==(const Date& d1) {
            if (d1.m_year == this->m_year && d1.m_month == this->m_month
                    && d1.m_day == this->m_day)
                return true;
            return false;
        }
};

void mystery(Date d1, Date d2) {
    Date d3 = d1;
    d1 = d2;
    d2 = d3;
}

int main(int argc, char** argv)
{
    Date diedler_birth(1988, 7, 15);
```

```
    Date paul_birth(1988, 8, 15);
    Date jacques_birth(1988, 8, 15);
    if (diedler_birth == jacques_birth) {
        std::cout << "Same birthday" << std::endl;
    }
    else {
        mystery(diedler_birth, paul_birth);
        diedler_birth.display();
        paul_birth.display();
        paul_birth.setMonth(7);

        if (diedler_birth == paul_birth)
            std::cout << "Yeah !" << std::endl;
        else std::cout << "Different birthday" << std::endl;
    }
    std::cout << "Bye !" << std::endl;
    return 0;
}
```

Ecrire la sortie du programme (ce qui sera affiché à l'écran lors de son exécution).

1pt par ligne correcte dans LE BON ORDRE

```
1988-7-15
1988-8-15
Yeah !
Bye !
```

Exercice 4 - Polymorphisme (5 points) / 1pt par question

Pour les questions 1, 2 et 3 on demande uniquement d'implémenter les interfaces (les fichiers headers) des classes Entity, Sword et Shield. Il faut donc définir les attributs, constructeurs surchargés et les destructeurs mais il n'est pas demandé de faire des accesseurs...

1) Ecrire une classe *Entity* contenant comme attributs deux positions *m_x* et *m_y* ainsi qu'une méthode *action* qui ne fait rien.

Mettre 0 si les attributs sont en publiques
-0.5pt si action est oubliée ou alors virtuelle
-0.5pt si constructeur surchargé oublié ou incorrect (constructeur par défaut n'est pas demandé !)
-0.5pt si destructeur oublié ou incorrect
-0.5pt si erreur de syntaxe

```
Class Entity
{
    Private: // ou protected
      Int m_x, m_y;
    Public:
      Entity(int _x, int _y); // constructeur surchargé
      ~Entity(); // destructeur

     void action(); // pas de virtual dans cette question !
};
```

2) Ecrire une classe *Sword* héritant de *Entity* ayant en plus comme attribut *m_power* (un réel)

Mettre 0 si l'attribut est en publique. Ne pas pénaliser si m_power est un entier...(au lieu d'un réel)

-0.5pt si héritage oublié ou incorrect

-0.5pt si constructeur surchargé oublié ou incorrect

-0.5pt si destructeur oublié ou incorrect

-0.5pt si erreur de syntaxe

```
Class Sword : public Entity
{
    Private:
      Float m_power;
    Public:
      Sword(float _power, int _x, int _y); // constructeur surchargé
      ~Sword(); // destructeur
};
```

3) Ecrire une classe *Shield* héritant de *Entity* ayant en plus comme attribut *m_defense* (un réel)

Même remarques que question précédente

```
Class Shield : public Entity
{
    Private:
      Float m_defense;
    Public:
      Shield(float _defense, int _x, int _y); // constructeur surchargé
      ~Shield(); // destructeur
};
```

On souhaite maintenant associer les actions suivantes pour nos deux classes filles :

- pour l'épée (*Sword*) : affichage du message "I love Zelda"
- pour le bouclier (*Shield*) : affichage du message "I can defense..."

4) Définir et implémenter (dans le fichier .cpp) les méthodes *action* pour les classes *Sword* et *Shield*.

Mettre 0 si "Sword::" ou "Shield::" est oublié !!

-0.5pt si erreur de syntaxe

```
// Dans le .H de Sword => void action();
Void Sword::action() {
   Std::cout << "I love Zelda" << std::endl;
}

// Dans le .h de Shield => void action();
Void Shield::action() {
  Std::cout << "I can defense" << std::endl;
}
```

On considère maintenant le *main* suivant :

```
int main(int argc, char** argv)
{
    std::vector<Entity*> entities;
    entities.push_back(new Sword(...));
    entities.push_back(new Shield(...));
    for (unsigned int i=0; i<entities.size(); i++)
    {
          // appel de la méthode action en fonction du type d'entité
          entities[i]->action();
    }
    return 0;
}
```

5) Faut-il modifier la classe mère *Entity* pour que ce programme fonctionne ? Si oui, que faut-il faire ?

Mettre 0 si la réponse est "OUI" sans justification ou avec justification fausse

-0.5pt si le virtual est oublié pour le destructeur de la classe Entity

-0.5pt si oubli de définir action() de la classe Entity virtuelle pure

Oui

il faut rajouter "virtual" devant le destructeur de la classe Entity
Et définir la méthode action() de Entity comme étant virtuelle pure !

Question bonus : Après l'application du polymorphisme, que devient la classe *Entity* de l'exercice précédent ? *(1 point)*

Mettre 0 si la réponse est "abstraite" avec n'importe quoi après !

1 point bonus pour abstraite sinon 0

Entity devient une classe abstraite, c'est à dire non instanciable

Bon courage !