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 (pas de points négatifs…)

Entourer la bonne réponse

Réponse A pour toutes les questions

```
Q1) Les attributs d'une classe sont :

        a) private
        b) public car cela respecte le principe d'encapsulation
        c) protected comme ca ils sont protégés
        d) les attributs ne sont pas dans les classes

Q2) Qu'est ce qu'un setter ?

        a) c'est une fonction qui permet de modifier la valeur d'un attribut
        b) c'est une fonction qui permet de lire la valeur d'un attribut
        c) c'est le constructeur
        d) c'est le destrucreur

Q3) Quelle est la syntaxe correcte pour les templates ?

        a) template <typename Foo, typename Bar>
        b) template<> <typename Foo>
        c) template (typename Foo)
        d) template<> <typename Foo, typename Bar>

Q4) Comment attraper une exception ?

        a) avec un bloc try / catch
        b) avec un bloc try / catch / finally
        c) avec un bloc if / else
        d) en courant après l'exception...

Q5) Qu'est ce qu'une table de hachage ?

        a) c'est un  tableau qui associe un identifiant à un objet
        b) c'est une liste chainée d'objets
        c) c'est un tableau qui hache les objets en petits morceaux
        d) c'est un tableau qui stocke des objets de type hétérogène
```

Exercice 1 : Modélisation et diagramme de classe (4 points)

On souhaite gérer une collection de *Personnage* de différents types. Pour cet exercice, on considérera uniquement les types suivants : *Mage*, *MageBlanc*, *MageNoir*, *Guerrier* et *Elfe*.

Les mages attaquent en lançant des sorts (prévoir juste un compteur de sort), les guerriers attaquent avec une ou deux épées et les elfes tirent de flèches.

Faire le diagramme de classe en faisant apparaitre pour chaque classe :

1. les attributs et leur visibilité
2. la méthode *attaquer*
3. les relations entre les classes

Il n'est pas demandé de faire apparaitre les constructeurs, destructeurs et getters/setters.

1) Diagramme de classe **(3 points)**

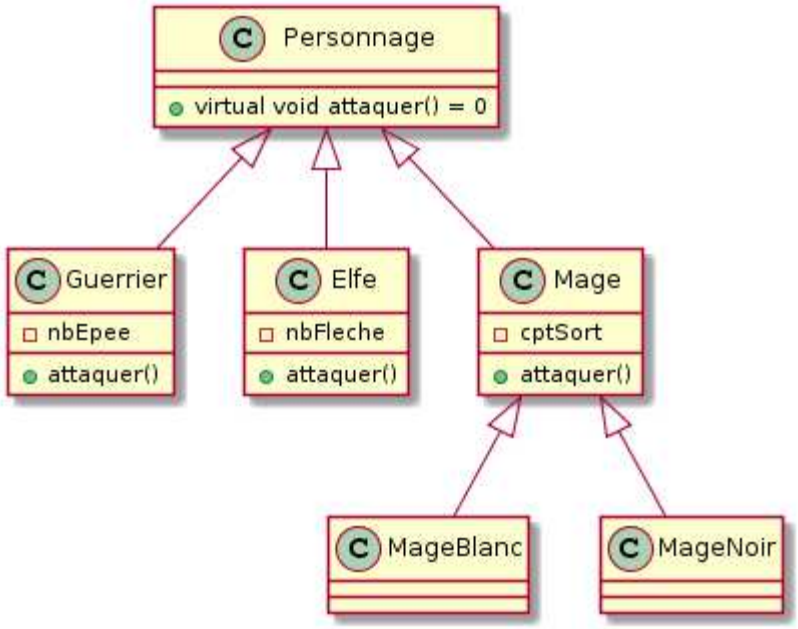
```
@startuml
class Personnage {
 +virtual void attaquer() = 0
}

class Guerrier{
 -nbEpee
 +attaquer()
}

class Elfe {
 -nbFleche
+attaquer()
}

class Mage{
 -cptSort
+attaquer()
}

Personnage <|-- Guerrier
Personnage <|-- Elfe
Personnage <|-- Mage
Mage <|-- MageBlanc
Mage <|-- MageNoir
@enduml
```



2) Expliciter en français la relation liant les classes *Personnage* et *MageNoir* **(1 point)**

Héritage à 2 niveaux : un MageNoir « est un » Mage qui « est un » Personnage
// -0,5 si juste marqué « héritage »

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

```
#include <vector>
#include <iostream>

void swap(std::vector<char>& tab, int a, int b) {
    char c = tab[a];
    tab[a] = tab[b];
    tab[b] = c;
}

void display(const std::vector<char>& tab) {
    for(unsigned int i = 0; i < tab.size(); i++) {
        std::cout << tab[i] << ",";
    }
    std::cout << "END" << std::endl;
}

void mystery(std::vector<char>& tab) {
    int min = 0;
    for(unsigned int i = 0; i < tab.size()-1; i++) {
        for(unsigned int j = i; j < tab.size(); j++) {
            if (j == i) min = j;
            if (tab[j] > tab[min]) min = j;
        }
        swap(tab, i, min);
        swap(tab, min, i);
    }
}

int main() {
    std::vector<char> tab = {'a','b', 'd', 'c'};
    mystery(tab);
    display(tab);
    return 0;
}
```

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

-1 point si pas de justifications !!

A,b,d,c,END
Il n'y a pas de swap car il est annulé par la ligne suivante : `swap(tab, min, i);`

Exercice 3 – Compréhension de code avancée (4 points)

```cpp
#include <vector>
#include <iostream>

template <typename T>
int mystery(T input, int magic = -10)
{
    return input + magic;
}

template<>
int mystery(float input, int magic)
{
    return (int)input;
}

template<>
int mystery(std::string input, int magic)
{
    return (int)input.size();
}
```

```cpp
int main()
{
    float a = 10.0f;
    std::string b = "hello";

    std::cout << mystery(10) << std::endl;
    std::cout << mystery(10.0) << std::endl;
    std::cout << mystery(a) << std::endl;
    std::cout << mystery(b) << std::endl;

    return 0;
}
```

Ecrire la sortie du programme (ce code compile bien !)

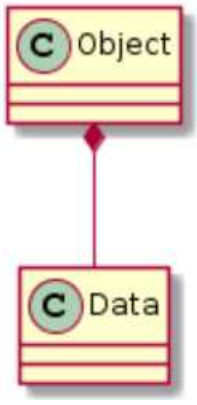
0
0
10 (appel de la 1ère spécialisation pour des float)
5 (appel de la 2ème spécialisation pour des string)

Exercice 4 – Composition et copie profonde (5 points)

On considère les classes suivantes (les attributs sont publiques par simplicité) :

```cpp
class Data
{
public:
    int m_value;
};

class Object
{
public:
    int m_a;
    Data* m_b;
};
```



Ecrire la fonction permettant de dupliquer un objet de type *Object* (4 points)

```
Object dupliquer(const Object& o)
{
   Object newObj ;
   newObj.m_a = o.m_a ;
   newObj.m_b = new Data() ; // très important le new ! -2 points si pas de new
   newObj .m_b->m_value = o.m_b->m_value ; // attention flèche car pointeur !

   return newObj;
}
```

Ecrire la ligne permettant d'appeler votre fonction *dupliquer* (1 point)

```
Object source;
Object clone = dupliquer(source) ; // -0,5 si l'étudiant écrit juste   « dupliquer(source) ; »
```

Bon courage !