**Question de cours (4 points) / 1pt par question**

1) Quelle est l'utilité des templates ?

|  |
| --- |
| Faire du code générique / modulaire en évitant de faire de la recopie de fonction/classe |

2) Citez deux problèmes liés à l'héritage multiple ?

|  |
| --- |
| Héritage en diamant : problème de conflits au niveau des appels des constructeurs, attributs portant le même nom, duplication en mémoire de l’objet de la classe de Base (celle en haut du dimant) |

3) A quoi sert le mot-clé *virtual* ?

|  |
| --- |
| A dire qu’une fonction sera polymorphe donc à faire du polymorphisme |

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.

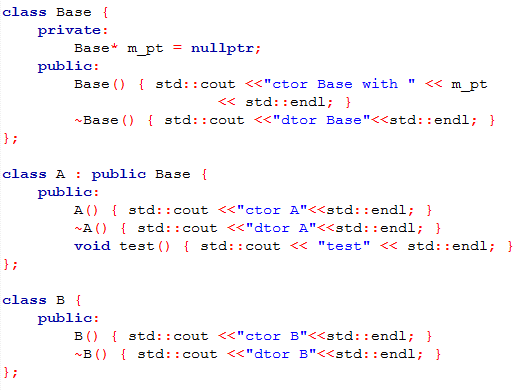
|  |
| --- |
| une pile car principe LIFO (Last In First Out)   ou un Set avec surcharge de l’opérateur < par ordre décroissant sinon ca ne marche pas avec celui par défaut qui trie par ordre croissant |

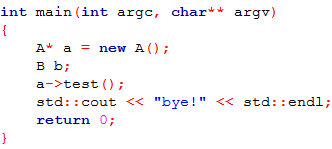
**Exercice 1 : Template simple (4 points)**

Ecrire une fonction permettant d'afficher le contenu d'une liste (std::list) valable pour les types int, char, float, double et std::string !

|  |
| --- |
| // noter au feeling  Template <typename T>  void afficher(const std ::list<T>& l) // accepter si passage par copie  {  For (const auto& elem : l) // accepter les variantes de parcours  {  Std ::cout << elem << std ::endl ;  }  } |

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

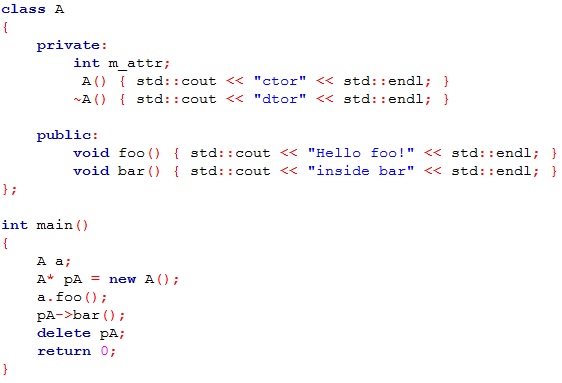




Ecrire la sortie du programme (ce qui sera affiché à l'écran lors de son exécution).

|  |
| --- |
| // 0,5pt par ligne correction et -0,5pt par ligne en trop  Ctor base with 0  Ctor A  Ctor B  Test  Bye  Dtor B  // NB : pas de dtor A, ni dtor Base car il n’y a pas le delete ! Enlever 1pt si l’étudiant a rajouté ces deux lignes |

**Exercice 3 - Compréhension de code (2 points)**



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

|  |
| --- |
| C’était la question piège : RIEN DU TOUT car le constructeur est en privé !  1pt pour la réponse et 1pt pour la justification !  Si l’étudiant a fait un trait oblique comme réponse, mettre juste 1pt  Si l’étudiant a laissé vide mettre évidemment 0pt |

**Exercice 4 - Polymorphisme (7 points) / 1pt par question**

**Pour les questions 1, 2 et 3 on demande uniquement d'implémenter les interfaces (les fichiers headers) des classes *Vehicle*, *EarthVehicle* et *WaterVehicle*. 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 *Vehicle* contenant comme attributs *m\_capacity (int)* et *m\_brand (string)* ainsi qu'une méthode *action* qui ne fait rien.

|  |
| --- |
| Class Vehicle {  Private :  Int m\_capacity ;  Std ::string m\_brand ;  Public :  Vehicle(int \_capacity, std ::string \_brand) ;  Void action() ;  } ; |

2) Ecrire une classe *EarthVehicle* héritant de *Vehicle* ayant en plus comme attribut *m\_wheels* (int)

|  |
| --- |
| Class EarthVehicle : public Vehicle {  Private :  Int m\_wheels;  Public :  EarthVehicle(int \_capacity, std ::string \_brand, int \_wheels) ;  } ; |

3) Ecrire une classe *WaterVehicle* héritant de *Vehicle*

|  |
| --- |
| Class WaterVehicle : public Vehicle {  Public :  WaterVehicle(int \_capacity, std ::string \_brand) ;  } ; |

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

* pour les véhicules terrestres (*EarthVehicle*) : affichage du message "I love earth"
* pour les véhicules aquatiques (*WaterVehicle*) : affichage du message "I hate water"

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

|  |
| --- |
| Void EarthVehicle ::action() { std ::cout << « …. » << std ::endl ; }  Void WaterVehicle ::action() { std ::cout << « …. » << std ::endl ; } |

On considère maintenant le *main* suivant :

**int** main(**int** argc, **char**\*\* argv)

{

std::vector<Vehicle\*> v;

v.push\_back(**new** EarthVehicle(...));

v.push\_back(**new** WaterVehicle(...));

v.push\_back(**new** EarthVehicle(...));

**for** (**unsigned** **int** i=0; i<v.size(); i++)

{

// appel de la méthode action en fonction du type d'entité

v[i]->action();

}

**return** 0;

}

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

|  |
| --- |
| Oui mettre le destructeur en virtual -> 0,5pt  Activer le polymorphisme sur la méthode action qui sera virtuelle **pure** -> 0,5pt |

6) Que peut-on dire de la classe *Vehicle* ? Justifiez !

|  |
| --- |
| Elle devient abstraite (0,5pt) ce qui veut dire qu’on ne peut pas instancier un objet de cette classe (0,5pt) |

7) Implémenter le constructeur surchargé de la classe *EarthVehicle*

|  |
| --- |
| EarthVehicle ::EarthVehicle(int \_capacity, std ::string \_brand, int \_wheels) : m\_capacity(\_capacity), m\_brand(\_brand), m\_wheels(\_wheels)  {  } |

Bon courage !