Question non répondue :

2.2 le [] delete

4.5

Partie 7

Partie 1 :

1. Décrire et comparer les macros du C et les fonctions (et méthodes) "inline" du C++. Toute fonction peut-elle être une "bonne" fonction inline ?

-il ne s’agit donc pas d’un appel de fonction au sens machine du terme

-N’a d’interet que si son code est de petite taille

-Tout fonction inline n’est pas forcement bonne car si elle est trop longue, cella va prendre plus de temps à compiler et augmenter la taille de l’exécutable. Ile faut trouver un équilibre entre les fonctions normal (appelles au sens machine) et les fonctions inline qui vont « copier-coller le code» a chaque appel de la fonction

2. Expliquer les divers avantages de l'utilisation des opérateurs new et delete par rapport aux fonctions malloc() et free(). Qu'en est-il en particulier des tableaux, des constructeurs et du destructeur ? [à compléter avec le cours 2]

-New et delete font des appels de constructeurs et destructeurs, on peut donc y exécuter du code et formater notre élément comme on le souhaite(ex : mettre un pointeur a nul)

-Malloc et free ne vont seulement allouer l’espace en mémoire et retourner un pointeur vers cette espace.

-Delete [] va delete tous les éléments du vecteur ou tableau, on ne dois pas parcourir chaque éléments

-Le constructeur est appelé pour chaque élément du tableau lors du new et le destructeur est appelé pour chaque élément lors du delete.

-Un objet est créé en utilisant l’opérateur new ; celui-ci appelle un constructeur qui permet d’initialiser l’objet

-Il est détruit en utilisant le l’opérateur delete ; celui-ci appelle le destructeur puis libère l’espace mémoire.

3. Expliquer le polymorphisme des fonctions et des opérateurs en C++. Pourquoi est-il possible alors qu'il ne l'est pas en C ?

-C++ permet `a une fonction d’avoir plusieurs versions différentes : on parle encore du polymorphisme des fonctions

-Une version d’une fonction diffère d’une autre version par le type d’au moins un paramètre. On parle encore de “signature” des fonctions

-Le type de la valeur renvoyée n’intervient pas du tout dans la distinction entre deux fonctions de même nom

-Le polymorphisme s’applique aussi aux opérateurs. Il est donc possible de surcharger de plusieurs manières un opérateur donné pour nos types de données, si cela a un sens.

4. Expliquer en quoi une classe de la P.O.O. en C++ apporte de la modularité et de l'abstraction. En particulier, quel est le rôle de l'encapsulation des variables membres en private accompagnée par des getters et setters en public.

-Pas trouvé dans le cours en private un variable ne peut être modifier que pas des fonctions membre de la classe, l’avantage est que l’utilisateur ne peut pas modifier la valeur comme il le veux mais dois passer par un setter ou getter pour modifier cette valeur. L’avantage est que l’on va tester si la valeur entre par l’utilisateur est correct ou non (c’est comme un blindage de saisie)

5. Expliquer ce qu'est le pointeur this et comment il intervient dans les définitions des méthodes d'une classe. Toute méthode possède-t-elle toujours ce paramètre ? [à compléter avec le cours 2]

-Chaque objet d’un même type (et donc instanciant une même classe) possède ses propres données mais partage le même code avec chaque objet de ce type. Comment le compilateur fait-il alors pour générer un code qui retrouve l’objet auquel les méthodes évoquées s’appliquent ? En fait, il y réussit car il passe un paramètre supplémentaire, et ce de manière implicite, à chaque fonction : c’est le pointeur this qui est un pointeur sur l’objet en question.

Partie 2 :

1. Décrire la forme typique du constructeur de copie d'une classe - en particulier, explique ce qu'est une référence en C++ et le rôle qu'elle joue dans le passage des paramètres d'une méthode ou d'une fonction. Expliquer les diverses situations où ce constructeur de copie est utilisé implicitement ou explicitement.

-Le constructeur de copie permet de créer un objet comme étant une copie d’un autre objet du même type. Un tel constructeur existe aussi de manière implicite. Il se contente d’affecter membre à membre les champs de l’objet copié à ceux du nouvel objet.

-Tout comme le constructeur par défaut, il convient d’écrire un constructeur de copie pour chaque classe développée (sous peine de gros souci… surtout en cas de variable membre de type pointeur)

Le constructeur de copie est appelé :

a) Lors de la création d’un objet à partir d’un autre du même type (appel explicite)

b) Lors du passage par valeur d’un objet dans une fonction (appel implicite)

c) Lorsqu’un objet est renvoyé par valeur par une fonction (appel implicite)

2. Expliquer ce qu'est un destructeur et comment il est appelé. Quel rapport avec l'opérateur delete [] (prendre un exemple) ?

-Le destructeur d’une classe est une fonction membre particulière. C’est la fonction qui est appelée lorsqu’on détruit l’objet (objet dynamique) ou lorsque l’on quitte sa portée (objet statique).

-Syntaxiquement, le destructeur est une fonction :

a) Qui porte le même nom que la classe précédée du symbole « ~ »

b) Qui n’a pas de type (même pas void)

c)Qui ne peut avoir de paramètres

d) Qui ne peut être surchargé : le destructeur d’une classe est unique

3. Expliquer et justifier les notions de variable membre statique et de méthode statique. Pourquoi peut-on imaginer des variables membres statiques dans une classe qui sont des instances de cette classe [à compléter cours 3]?

-Le but d’une variable statique est d’être unique (en mémoire (pour prendre moins de place)) entre toute les instanciation de la même classe.

-La variable statique est donc une sorte de variable globale qui existe tout au long du programme -> elle est donc accédée directement, c’est à-dire sans passer par le pointeur this d’un objet de la classe

- L’accès à la variable statique publique : on utilise l’opérateur de résolution de portée « :: » avec le nom de la classe

- On peut utiliser cette variable statique avant même qu’un seul objet de cette classe n’existe

4. Expliquer et illustrer les différentes associations possibles entre deux classes [à compléter avec le cours 4]. En particulier, comment un objet d'une classe peut-il "envoyer un message" à un objet instance de l'autre classe ? [à compléter avec cours 3]

-L’operateur friend : Une fonction amie d’une classe possède les mêmes privilèges vis-à-vis de celleci que ses fonctions membres. Elle peut donc accéder à sa partie privée. Il faut donc être prudent dans la création de telles fonctions afin de ne pas remettre en cause le principe de l’encapsulation.

-L’heritage : Si un classe B hérite de la classe A, elle aura accès à toutes ces méthode et variables membre public mais pas private ! Il faut changer le private ne protected pour y avoir accès en héritage.

Partie 3 :

1. Qu'est-ce qu'un opérateur membre et un opérateur ami ? Dans quelles circonstances choisit-on l'un ou l'autre pour un opérateur donné ? Donner et expliquer un exemple pour un opérateur binaire de cas où ♦ la 1ère méthode est obligatoire, ♦ la 2ème méthode est obligatoire, ♦ on peut utiliser indifféremment l'une ou l'autre méthode.

-Un opérateur membre assure que le premier opérande est forcément un objet de la classe : aucune conversion n’est possible.

- Au contraire, un opérateur libre (ou ami) a pour première opérande un argument de fonction normal. Il peut donc être quelconque.

- Un opérateur libre est souvent utilisé lorsque les opérandes sont des objets de type différent, qui peuvent se présenter dans n’importe quel ordre

2. Justifier et expliquer les formes classiques des surcharges des opérateurs +, >, << et ++.

J’ai pris imageNG comme exemple

- ImageNG operator++() pré incrémentation

-ImageNG operator++(int) post incrémentation

-ImageNG& operator=(const ImageNG&);

-ImageNG operator+(int);

-bool operator>(const ImageNG&);

-friend ostream& operator<<(ostream&,const ImageNG&);

3. Expliquer en utilisant un exemple pourquoi il peut arriver que l'on déclare une méthode en private.

-Si l’on souhaite que la méthode ne soit accessible que par l’objet en lui-même (ces méthodes) et donc que l’utilisateur n’y ait pas accès. C’est une protection pour que l’utilisateur ne fasse de la merde.

4. Expliquer comment certains appels implicites d'un constructeur ou d'un casting peuvent conduire à une application s'exécutant sans erreurs mais donnant des résultats erronés. Comment se prémunir de ce genre de problème ?

-En l’absence de constructeur au sein de la classe dérivée, celle-ci hérite, implicitement, du constructeur par défaut et du constructeur de copie de la classe de base, pas des constructeurs d’initialisation.

-Pour régler ce problème, il faut appeler explicitement le constructeur d’initialisation de la classe mère

CclassFille (int a, int b) : CclassMere()

{

classFille.a = a ;

classFille.b = b ;

}

Partie 4 :

1. Expliquer et illustrer les différentes associations possibles entre deux classes au moyen des classes Repas, Plat, Banquet, Ingredients, CalculPrix, RepasBanquet, …. Donner le code de base de chaque classe avec l'implémentation concrète de l'association. Comment faire

-Agrégation par valeur ou référence :

-Par valeur : Le plat fait partie du repas. Sans repas, le plat n’existe pas. Le plat est donc une partie du repas et ne peux exister sans lui. Le plat seul ne sert à rien. Il n’a donc pas de raison d’exister sans repas. Si le repas est détruit, le plat l’est avec lui. (plat peut donc être abstract et non instanciable sans repas ???)

-Par référence : Les ingrédients servent à préparer un ou plusieurs plats. Ils ne sont donc pas dépendant d’un plat. Ils peuvent donc exister sans un plat. Le plat utilise certain ingrédient mais les ingrédients sont utilisés par différents plats. Ils n’appartiennent donc pas aux plats et peuvent exister sans eux. Si le plat est détruit, les ingrédients ne le sont pas.

2. Dans quelle situation peut-on être amené à créer un héritage ? Comment redéfinir une méthode ou un constructeur dans une classe dérivée en utilisant la méthode ou un constructeur de la classe mère existant ?

- Il vise la récupération du code développé pour un type d’objet donné en vue d’en créer un nouveau type semblable au premier mais en le spécialisant davantage ou l’étendant au niveau de ses données et de ses fonctionnalités.

-Pour redéfinir une méthode, il suffit de la copier dans le .h et dans le .cpp ensuite on peu la modifier comme on le souhaite dans le .cpp

ImageNG::ImageNG() : Image()

Le : Image() fait appelle au constructeur de la classe mère

3. Dans le cas d'une redéfinition de méthode au sein d'un processus d'héritage, décrire exhaustivement (avec code minimal) les deux cas où il faut obligatoirement les qualifier de virtuel ? Comment faut -il appeler ces méthodes pour que le mécanisme fonctionne ?

Ne pas oublier : Une méthode virtuelle est une méthode dont la résolution de l’appel s’effectue à l’exécution et pas lors de la compilation du programme.

-Si on appel cette méthode via un pointeur, il faut absolument la redéfinir virtuelle. Sinon, ce n’est pas nécessaire. Si elle n’est pas virtuelle et qu’on utilise un pointeur, on appellera toujours la méthode de la classe mère

-Le destructeur doit toujours être virtuel

4. Expliquer par quels mécanismes il est possible, dans un processus d'héritage 1) d'assurer que méthode originale ou la méthode redéfinie sera appelée au sein d'un ensemble mixte d'objet de base et d'objets dérivés 2) de pouvoir utiliser de manière sûre une méthode spécifique aux objets dérivés au sein d'un ensemble mixte d'objet de base et d'objets dérivés.

-Grace au virtuelle car Une méthode virtuelle est une méthode dont la résolution de l’appel s’effectue à l’exécution et pas lors de la compilation du programme.

5. Expliquer les manières d'utiliser les éléments d'une librairie orientée objet en prenant comme exemple une librairie O.O. d'interfaces graphiques Expliquer sur base d'un exemple concret pourquoi le downcasting est parfois nécessaire et quel opérateur "safe" utiliser.

-Pas trop compris, il dois parler de quand on fais on dynamic\_cast d’une imageRGB\* vers une image\*. Le dynamic\_cast est considere comme safe dans ce cas appart ca jsp

6. Expliquer dans quelles circonstances il peut être intéressant de définir en virtual le destructeur. Même question pour une méthode inline. Décrire la structure et le fonctionnement du code particulier mis en place dans ce cas par le compilateur.

- Une méthode virtuelle est une méthode dont la résolution de l’appel s’effectue à l’exécution et pas lors de la compilation du programme.

-Lorsqu’on va appeler le destructeur d’une ImageRGB, le constructeur de Image dois être virtuel pour que lors de l’exécution on choisis le bon destructeur

Partie 5 :

1. Expliquer le fonctionnement de l'héritage multiple en C++; en particulier, expliquer le problème qui peut survenir et comment le résoudre. Même question pour l'héritage en diamant.

-Ex avec l’héritage ne diamant on dois donc faire de l’héritage virtuel pour éviter de multiplier plusieurs fois la même class ainsi que toute ces variables membres.

-Info : Le mécanisme d’héritage virtuel impose d’appeler explicitement le constructeur de la super classe. Dès qu’une classe de base est virtuelle, ceci est valable pour toutes les classes de la hiérarchie qui dépendent d’elle.

-Voir fin du slide 4 pour plus d’info

2. Expliquer le fonctionnement du lancement d'une exception et le comparer au mécanisme C du longjump. En particulier, expliquer comment une instance de classe d'exception peut être récupérée pour gérer l'erreur survenue.

- Lors de l’utilisation d’un jump, le système enregistre l’environnement dans lequel il se trouve pour y retourner lorsque c’est nécessaire. Le try fait la même chose mais supprime les objets créés depuis l’entré dans le bloc try si le catch se déclenche.

-Le rôle d’une classe d’exception est de gérer et normaliser les erreurs survenant lors de l’exécution du programme. Celle-ci sont prévisibles et donc traitable.

3. Qu'est-ce qu'une classe d'exception ? Quels éléments comporte-t-elle au minimum et pourquoi ? Quelle est la structure classique d'un handler d'exception ?

- class BaseException

{

protected:

char \*message;

public:

BaseException(void);

BaseException(const char\* newMessage);

BaseException(const BaseException&newException);

~BaseException(void);

void setMessage(const char\*newMessage);

const char\* getMessage(void) const;

};

4. Expliquer et justifier ce que l'on entend par "container" en C++ en vous basant sur l'exemple de la classe Vecteur. En particulier, en utilisant l'exemple de la classe Pile, expliquer comment l'encapsulation appliquée dans les classes containers assure le maximum d'abstraction.

- Un container (conteneur en français) est une classe qui contient une collection de données (objets ou types de base).

-La pile hérite du vecteur de base mais va modifier les méthode d’ajout et de retrait pour correspondre a ses choix de gestion des données.

Partie 6 :

1. Définir ce qu'est une classe abstraite et expliquer les rôles en utilisant l'exemple des containers liste/pile/pile/listeTriee. Expliquer et comparer la manière utilisée en C++ et en Java pour indiquer qu'une classe est effectivement abstraite ; quel effet souhaite-t-on ainsi obtenir ?

-Une classe abstraite est une classe qu’on ne peu pas instancier mais elle est utile pour les classe qui vont en hérité(Ex : Image ne sert à rien sans ImageNG ou ImageRGB etc)

-Pour qu’une classe soit abstraite en C++ on doit mettre une de des méthode a = 0 ;

-En java on doit simplement écrire abstract avant la déclaration de la classe

2. Expliquer et justifier la notion d'itérateur : quel est l'objectif poursuivi ? Quel est le code C++ classique de parcours d'un container au moyen d'un itérateur ? Quelles sont typiquement les variables membres et les méthodes que l'on trouve dans ce genre de classe ? Illustrer avec le code de base de l'opérateur ++ pour un itérateur de vecteur et de liste chaînée.

- Un itérateur est un objet qui permet de balayer une structure de données, et d’accéder ainsi à tous les objets qu’elle contient, selon un ordre bien établi.

-L’Object est d’avoir un objet simple pour en parcourir un autre plus complexe (Ex : pour parcourir une List Triée)

-On trouve une référence sur l’Object à parcourir et la position a laquelle l’itérateur se trouve

3. Expliquer ce qu'est une classe "template" (ou générique" ou "paramétrable"). Illustrer et justifier, avec le code d'un vecteur template basique, la syntaxe utilisée dans différentes situations.

-La notion de template permet de créer en C++ des classes (ou des fonctions) génériques, c’est-à-dire indépendantes d’un type de données.

Instanciation des templates :

-Evidemment, puisque ces classes (fonctions) sont génériques, le compilateur ne crée pas de code lorsqu’on se contente de les définir. Il le fera lorsque l’on précisera quelle version de la classe (ou de la fonction) on désire utiliser dans notre programme.

Partie 7 :

1. Expliquer le problème de l'instanciation des templates dans le cas où l'on travaille en modules séparés (pourquoi/comment il faut-il "forcer" cette instanciation ?). Illustrer par l'exemple d'un vecteur template utilisé pour manipuler trois vecteurs distincts (par exemple, un vecteur d'entiers, un vecteur de Cdate et un vecteur de char).

2. Expliquer en la justifiant l'architecture des flux en 2 (4) couches du C++. Un flux de bas niveau est-il utilisable sans un flux de haut niveau associé ? Et l'inverse ? Décrire (pas seulement citer !) les principales classes (et leurs associations d'héritage et d'agrégation) nécessaires pour réaliser les traitements standards sur des fichiers.

3. Expliquer ce qu'est un manipulateur dans le contexte des flux et expliquer la surcharge de l'opérateur << qui justifie la manière de l'utiliser. Décrire les manipulateurs endl, ends et quelques manipulateurs de formatage.

4. Expliquer comment on procède en pratique pour écrire des données dans un fichier binaire (méthode avec deux classes, méthode avec une seule classe) ou dans un fichier texte. En quoi les deux fichiers obtenus sont-ils différents ?

5. Expliquer à partir d'un exemple pourquoi il est intéressant de travailler sur une zone mémoire en utilisant un flux.

6. Expliquer ce qu'est l'objet cout et ses fonctionnalités en termes d'écriture formatée, de surcharges de l'opérateur << et de redirection.