Nom Prénom : STROUSTRUP Bjarne	Groupe : C++

R3.04 / QCM / Durée 36 mn / Tous Documents Autorisés

Ecrivez votre Nom, Prénom et Groupe en haut de **chaque** page

Cochez **sans ambiguïté** les bonnes réponses : ☒

- Dans le code qui est proposé, on supposera toujours que toutes les directives `include` nécessaires ont bien été prévues, ainsi que « `using namespace std ;` ». Aucune erreur ne peut provenir de cela.
- Pour limiter la place occupée par le code, on implémente les méthodes directement dans leur spécification (.h). Cela fonctionne, même si bien sûr c'est à éviter lorsque l'on développe !

Question 1. Classes, Héritage, Constructeur, Destructeur, Allocation

On fournit les deux classes suivantes pour représenter une **Personne** et un **Contribuable**, qui est une sorte de **Personne**. Lisez attentivement ce code avant de répondre aux questions.

```
class Personne {
public:
  Personne(const string & nom = "Lambda"): m_nom(nom) {
    cout << "Const. Personne " << m_nom << endl;
  }
  ~Personne() { cout << "Destr. Personne " << m_nom << endl; }
private:
  string m_nom;
};
```

```
class Contribuable : public Personne {
public:
  Contribuable(float impot) : Personne(), m_impot(impot) {
    cout << "Const. Contribuable " << m_impot << "€" << endl;
  }
  ~Contribuable() { cout << "Destr. Contribuable " << m_impot << "€" << endl; }
private:
  float m_impot;
};
```

Question 1.1. Combien de **méthodes** possède un objet de la classe **Contribuable ?**
(on compte toutes les méthodes y compris constructeur et destructeur)

A☐ Un	B☐ Deux	C☐ Trois	D☒ Quatre	E☐ Cinq	F☐ Six	G☐ Sept

Question 1.2. Combien **d'attributs** possède un objet de la classe **Contribuable** ?

A☐ Un	B☒ Deux	C☐ Trois	D☐ Quatre	E☐ Cinq	F☐ Six	G☐ Sept

Question 1.3. Soit le programme principal :

```
int main() {
  Personne     unePersonne("Martin");
  Contribuable unContribuable(1000);
  return 0;
}
```

On vous rappelle qu'en C++ :

- *Les variables sont allouées dans l'ordre dans lequel elles sont déclarées*
- *Les variables sont désallouées dans l'ordre inverse de leur déclaration*
- *Les destructeurs sont appelés dans l'ordre inverse des constructeurs*

Lors de l'exécution de ce programme, quelle sera la trace dans la console ?

A☐	B☒	C☐	D☐

(Rien ne s'affiche)	`Const. Personne Martin`
	`Const. Personne Lambda`
	`Const. Contribuable 1000€`
	`Destr. Contribuable 1000€`
	`Destr. Personne Lambda`
	`Destr. Personne Martin`
		`Const. Personne Martin`
		`Const. Contribuable 1000€`
		`Destr. Contribuable 1000€`
		`Destr. Personne Martin`
			`Const. Personne Martin`
			`Const. Personne Lambda`
			`Const. Contribuable 1000€`

Nom Prénom : STROUSTRUP Bjarne	Groupe : C++

? Question 1.4. Soit le programme principal :

```
int main() {
  Personne *     unePersonne = new Personne("Martin");
  Contribuable * unContribuable = new Contribuable(1000);
  return 0;
}
```

Lors de l'exécution de ce programme, quelle sera la trace dans la console ?

A☐	B☐	C☐	D☒
(Rien ne s'affiche)	Const. Personne Martin Const. Personne Lambda Const. Contribuable 1000€ Destr. Contribuable 1000€ Destr. Personne Lambda Destr. Personne Martin	Const. Personne Martin Const. Contribuable 1000€ Destr. Contribuable 1000€ Destr. Personne Martin	Const. Personne Martin Const. Personne Lambda Const. Contribuable 1000€

Question 2. Manipulation de pointeurs et références

On nous fournit une classe permettant de représenter une `Image`. Cette classe comporte un constructeur par défaut et une méthode d'instance qui permet de superposer à l'image `*this` le contenu d'une autre `Image`. La spécification de la méthode `superposer` est la suivante :

```
void Image::superposer (const & Image autreImage);
```

? Question 2.1. Allocation Dynamique

On a alloué dynamiquement deux objets `Image` :

```
Image * im1 = new Image;  Image * im2 = new Image;
```

Parmi les instructions suivantes, laquelle ou lesquelles sont syntaxiquement correctes ?

A☐	`im1.superposer(im2);`
B☒	`(*im1).superposer(*im2);`
C☐	`(*im1).superposer(&im2);`
D☐	`im1->superposer(&im2);`
E☒	`(*(&im1))->superposer(*im2);`
F☐	`im1->superposer(im2);`

? Question 2.2. Allocation Automatique

On a alloué automatiquement 2 objets `Image` (dont un manipulé *via* une référence) :

```
Image im0;  Image & im1 = im0;  Image im2;
```

Parmi les instructions suivantes, laquelle ou lesquelles sont syntaxiquement correctes ?

A☐	`im1->superposer(&im2);`
B☐	`im1.superposer(&im2);`
C☒	`(*(&im1)).superposer(im2);`
D☐	`(*im1).superposer(*im2);`
E☒	`im1.superposer(im2);`
F☐	`im1->superposer(im2);`

Nom Prénom : STROUSTRUP Bjarne	Groupe : C++

Question 3. Spécifier

Soit une classe `Matrice` pour représenter une matrice 3x3. On demande d'écrire la spécification d'une procédure ou d'une fonction `somme` qui doit calculer la somme de deux matrices.

Question 3. Parmi les propositions suivantes, laquelle ou lesquelles permettront de réaliser le travail demandé ? *(cocher une solution si elle fonctionne, peu importe si elle n'est pas efficace)*

A ☒	`Matrice somme(Matrice m1, Matrice m2);`
B ☒	`const Matrice somme(const Matrice m1, const Matrice m2);`
C ☐	`void somme(Matrice & m1, Matrice & m2, const Matrice & mResultat);`
D ☒	`void somme(Matrice m1, Matrice m2, Matrice & mResultat);`
E ☐	`void somme(const Matrice & v1, const Matrice & v2, const Matrice & vResultat);`

Question 4. Spécifier Encore

On propose trois spécifications d'une fonction permettant d'ajouter deux objets de type `Vecteur` pour produire un autre objet `Vecteur`.

Question 4. Les quatre spécifications suivantes sont correctes, mais laquelle est la meilleure en termes de spécification et d'efficacité à l'exécution ?

A ☐	`Vecteur ajouter(const Vecteur v1, const Vecteur v2);`
B ☐	`Vecteur ajouter(Vecteur v1, Vecteur v2);`
C ☐	`Vecteur ajouter(Vecteur & v1, Vecteur & v2);`
D ☒	`Vecteur ajouter(const Vecteur & v1, const Vecteur & v2);`

Question 5. Spécifier Toujours

On a développé deux classes : `Livre` et `Page`. Chaque `Livre` est composé de plusieurs `Pages`. La spécification de la classe `Livre` est la suivante :

```
class Livre {
public :
  /* ici viendra la spécification de getPages */
private:
  vector<Page> m_pages;
};
```

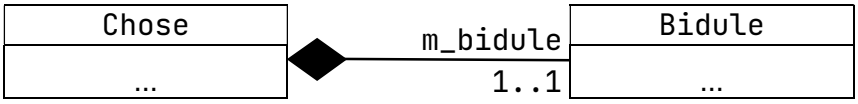
Question 5. Dans la classe `Livre`, quelle est la meilleure spécification pour la méthode `getPages` permettant de consulter les `Pages` qui composent un `Livre` ?

A ☐	`const vector<Page> & getPages();`	D ☐	`vector<Page> & getPages() const;`
B ☒	`const vector<Page> & getPages() const;`	E ☐	`vector<Page> getPages();`
C ☐	`vector<Page> & getPages();`	F ☐	`const vector<Page> getPages() const;`

Nom Prénom : STROUSTRUP Bjarne	Groupe : C++

Question 6. Agrégation ou Composition ?

On considère la spécification UML suivante :

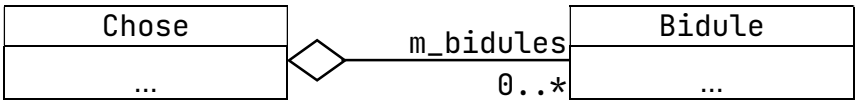


Question 6. Dans la classe `Chose`, parmi les propositions suivantes, cochez les types que pourrait avoir l'attribut `m_bidule` pour traduire correctement ce schéma UML :

A ☐	`Bidule *`	D ☐	`const Bidule *`
B ☐	`Bidule &`	E ☐	`const Bidule &`
C ☒	`Bidule`	F ☐	`Bidule * const`

Question 7. Agrégation ou Composition ?

On considère la spécification UML suivante :



Question 7. Dans la classe `Chose`, parmi les propositions suivantes, cochez les types que pourrait avoir l'attribut `m_bidules` pour traduire correctement ce schéma UML :

A ☒	`vector<Bidule *>`	D ☒	`vector<const Bidule *>`
B ☐	`vector<Bidule &>`	E ☐	`vector<const Bidule &>`
C ☐	`vector<Bidule>`	F ☐	`vector<const Bidule>`

Question 8. Coplien, Destructeur

Soit une classe `Livre` et une classe `Page`. Un `Livre` est composé, ou pas, de plusieurs `Pages`, chacune avec un numéro. On propose le début de code suivant :

```
class Page {
public : Page(unsigned int numPage) : m_numPage(numPage) {}
private: unsigned int m_numPage;
};

class Livre {
public:
    Livre(unsigned int nombrePages) {
        if (nombrePages > 0) {
            m_contenu = new vector<Page>;
            for(unsigned int i=0; i < nombrePages; i++) {
                m_contenu->push_back(Page(i)); /* création pages */
            }
        } else { m_contenu = nullptr; }
    }
    virtual ~Livre() { /* implémentation ici du destructeur */ }
private:
    vector<Page> * m_contenu;
};
```

Nom Prénom : STROUSTRUP Bjarne	Groupe : C++

Question 8. Parmi les implémentations du destructeur **~Livre()** proposées ci-dessous, laquelle ou lesquelles feront bien ce que doit faire le destructeur d'une telle classe ?

A☐	`/* le destructeur n'a rien à faire */`
B☒	`delete m_contenu;`
C☐	`m_contenu->clear();`
D☐	`if(m_contenu!=nullptr) { for(auto & page : *m_contenu) delete & page; }`
E☐	`while(!m_contenu->empty()) m_contenu->pop_back();`
F☐	`delete this;`

Nota Bene : *- la méthode `pop_back()` supprime le **dernier** élément d'un `vector`*
*- la méthode `clear()` supprime **tous** les éléments d'un `vector`*
- le template de classe vector est évidemment sous forme canonique de Coplien

Question 9. Héritage, Polymorphisme

On considère les trois classes suivantes : `Animal`, `Quadrupede` (qui hérite de `Animal`) et `Chien` (qui hérite de `Quadrupede`). Ces 3 classes sont spécifiées et implémentées de la manière suivante :

```
class Animal {
public :
  Animal(const string & race="Animal") : m_race(race) {}
  void afficherCri() const             {cout << "Grrrrrr !" << endl;};
  virtual void afficherRace() const    = 0 ;
protected :
  string m_race;
};
```

```
class Quadrupede : public Animal {
public :
 Quadrupede(const string & race="Quadrupede") : Animal(race) {}
 virtual void afficherCri() const             {cout << "Au Galop !" << endl; };
 void afficherRace() const override           {cout << m_race << endl; }
};
```

```
class Chien : public Quadrupede {
public :
 Chien(const string & race = "Chien") : Quadrupede(race) {}
 void afficherCri() const override    { cout << "Whou Whou !" << endl; };
};
```

Les classes ont été compilées sans erreur. Après les avoir bien lues, répondez aux questions :

Question 9.1. Soit le programme :

```
int main() { Animal unAnimal; unAnimal.afficherCri(); return 0 ;}
```

Que se passe-t-il si l'on compile et exécute ce programme ?

A☒	Il y a une erreur à la compilation du programme principal
B☐	Le programme compile, s'exécute et plante lors de l'exécution
C☐	Le programme compile, s'exécute et affiche « `Grrrrrr !` »

Nom Prénom : STROUSTRUP Bjarne	Groupe : C++

Question 9.2. Soit le programme :

```
int main() { Animal unAnimal; unAnimal.afficherRace(); return 0 ;}
```

Que se passe-t-il si l'on compile et exécute ce programme ?

A☒	Il y a une erreur à la compilation du programme principal
B☐	Le programme compile, s'exécute et plante lors de l'exécution
C☐	Le programme compile, s'exécute et affiche « Grrrrrr ! »

Question 9.3. Soit le programme :

```
int main() {
  Animal* chien = new Chien; chien->afficherCri(); return 0;
}
```

Que se passe-t-il si l'on compile et exécute ce programme ?

A☒	Le programme compile, s'exécute et affiche « Grrrrrr !»
B☐	Le programme compile, s'exécute et affiche « Au Galop !»
C☐	Le programme compile, s'exécute et affiche « Whou Whou !»

Question 9.4. Soit le programme :

```
int main() {
  Animal* chien = new Chien; chien->afficherRace(); return 0;
}
```

Que se passe-t-il si l'on compile et exécute ce programme ?

A☐	Le programme compile, s'exécute et affiche « Animal »
B☐	Le programme compile, s'exécute et affiche « Quadrupede »
C☒	Le programme compile, s'exécute et affiche « Chien »

Question 9.5. Soit le programme :

```
int main() {
  Chien chien; Quadrupede & quadru=chien; quadru.afficherCri(); return 0;
}
```

Que se passe-t-il si l'on compile et exécute ce programme ?

A☐	Le programme compile, s'exécute et affiche « Grrrrrr !»
B☐	Le programme compile, s'exécute et affiche « Au Galop !»
C☒	Le programme compile, s'exécute et affiche « Whou Whou !»

Question 9.6. Soit le programme :

```
int main() {
  Chien chien; Quadrupede & quadru=chien; quadru.afficherRace(); return 0;
}
```

Que se passe-t-il si l'on compile et exécute ce programme ?

A☐	Le programme compile, s'exécute et affiche « Animal »
B☐	Le programme compile, s'exécute et affiche « Quadrupede »
C☒	Le programme compile, s'exécute et affiche « Chien »