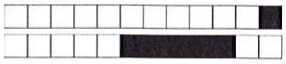


+1/1/60+

QCM **M1if02 Programmation Avancée, Université Lyon 1**

Test
Examen du 06/10/2017

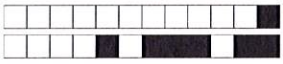
Nom Prénom et Numéro d'Etudiant:

.....

Durée : 1h30 minutes.
Aucun document n'est autorisé. L'usage de la calculatrice et du téléphone portable est interdit.
Les questions faisant apparaître le symbole ♣ peuvent présenter zéro, une ou plusieurs bonnes réponses. Les autres ont une unique bonne réponse.
Des points négatifs pourront être affectés à de mauvaises *réponses.*
Le langage utilisé dans cette UE et dans ce QCM est le C++. Il est néanmoins nécessaire d'avoir du recul sur d'autres langages comme Java.

Question 1 ♣ A propos de la mise en œuvre du polymorphisme en `C`, `C++` et `Java` :

- ☐ En C la mise en œuvre du polymorphisme sur les objets a un coût
- ☐ En C++ la mise en œuvre du polymorphisme sur les objets est automatique
- ☐ En Java la mise en œuvre du polymorphisme sur les objets a un coût
- ☐ En C++ la mise en œuvre du polymorphisme sur les objets a un coût
- ☐ En C la mise en œuvre du polymorphisme sur les objets est automatique
- ☐ En Java la mise en œuvre du polymorphisme sur les objets n'a pas de coût
- ☐ En C la mise en œuvre du polymorphisme sur les objets n'a pas de coût
- ☐ En Java la mise en œuvre du polymorphisme sur les objets est automatique
- ☐ En C++ la mise en œuvre du polymorphisme sur les objets n'a pas de coût
- ☐ *Aucune de ces réponses n'est correcte.*



Question 2 Qu'affiche le programme suivant?

```
struct LaBase{
    virtual void afficheType() {std::cout << LaBase << std::endl;}
};

template <typename Derivee>
struct LaSpecialisee : public LaBase{
    virtual void afficheType() {std::cout << typeid(Derivee).name() << std::endl;}
};

struct LaDerivee : public LaSpecialisee<LaDerivee> {};

int main() {
    LaBase *baby= new LaDerivee;
    baby->afficheType();
   return 0;
}
```

- ☐ LaDerivee
- ☐ Il y a une erreur à la compilation
- ☐ LaBase
- ☐ LaSpecialisee<LaDerivee>

Question 3 Qu'est-ce que `::new`?

- ☐ Il s'agit d'une directive de compilation
- ☐ C'est l'opérateur d'allocation global
- ☐ C'est l'opérateur d'allocation local d'une classe, dans la portée de laquelle se trouve cet appel
- ☐ Il s'agit d'une erreur de syntaxe

Question 4 La désallocation de la mémoire allouée par un programme se fait :

- ☐ Automatiquement pour les objets alloués avec l'opérateur `new` et explicitement pour les autres
- ☐ Explicitement pour tous les objets alloués avec l'opérateur `new` et automatiquement pour les autres
- ☐ Automatiquement pour tous les objets alloués
- ☐ Explicitement pour tous les objets alloués

Question 5 Lorsque l'on souhaite créer un pointeur pour pointer sur un objet de type `LaClasse` présent dans la pile, voici comment il faut procéder :

```
// Manière 1
LaClasse lala;
LaClasse *p = new LaClasse;
p=&lala;

// Manière 2
LaClasse lala;
LaClasse *p = new LaClasse;
p=lala;

// Manière 3
LaClasse lala;
LaClasse *p = new LaClasse;
*p=lala;

// Manière 4
LaClasse lala;
LaClasse *p = new LaClasse(lala);

// Manière 5
LaClasse lala;
LaClasse p = new LaClasse;
p=*lala;

// Manière 6
LaClasse lala;
LaClasse *p;
p=&lala;

// Manière 7
LaClasse lala;
LaClasse *p;
*p=lala;
```

- ☐ Manière 3
- ☐ Manière 4
- ☐ Manière 1
- ☐ Manière 6
- ☐ Manière 2
- ☐ Manière 5
- ☐ Manière 7

Question 6 Qu'affiche le programme suivant?

```
#include <iostream>
template <int i>
void procedure(){
   i += 2;
   std::cout << i;
}

int main(){
   procedure<5>();
   return 0;
}
```

- ☐ Il y a une erreur à la compilation
- ☐ 5
- ☐ 2
- ☐ 7

Question 7 Que fait `delete ptr`?

- ☐ Restitue l'espace mémoire occupé par `ptr`
- ☐ Cela dépend du type de la donnée pointée
- ☐ Restitue l'espace mémoire occupé par la donnée pointée par `ptr`

Question 8 Qu'affiche le programme suivant?

```
class LaBase{
public:
  virtual void affiche() {std::cout << "LaBase" << std::endl;}
};

class LaDerivee : public LaBase{
public:
  virtual void affiche() {std::cout << "LaDerivee" << std::endl;}
};

int main(){
   LaDerivee  dede;
   LaBase & bibi = dede;
   bibi.affiche();
   return 0;
}
```

- ☐ LaBase
- ☐ Il y a une erreur à la compilation
- ☐ LaDerivee

Question 9 ♣ On considère une classe `LaClasse` (non décrite ici), des fonctions permettant de manipuler des objets de cette classe, ainsi que des instances.

```
LaClasse f1(LaClasse &x)
{return x;}

LaClasse & f2(LaClasse &x)
{return x;}

LaClasse && f3(LaClasse &x)
{return std::move(x);}

LaClasse a;
LaClasse &b=a;
LaClasse tab[8];
```

Quelles affirmations sont correctes?

- ☐ `a` est une *lvalue*
- ☐ `f3(a)` est une *lvalue*
- ☐ `f2(a)` est une *lvalue*
- ☐ `f2(a)` est une *xvalue*
- ☐ `b` est une *xvalue*
- ☐ `f1(a)` est une *xvalue*
- ☐ `f1(a)` est une *lvalue*
- ☐ `a` est une *xvalue*
- ☐ `b` est une *lvalue*
- ☐ `tab` est une *lvalue*
- ☐ `f3(a)` est une *xvalue*
- ☐ `tab` est une *xvalue*
- ☐ *Aucune de ces réponses n'est correcte.*

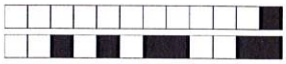
Question 10 Qu'affiche le programme suivant?

```
#include <iostream>

template <typename T>
void procedure(const T&x){
    static int compteur = 1;
    std::cout <<  x << " : compteur = " << compteur << " ; ";
    ++compteur;
    return;
}

int main(){
    procedure(1);
    procedure(5);
    procedure(5.5);
    return 0;
}
```

- ☐ 1 : compteur = 1 ; 5 : compteur = 1 ; 5.5 compteur = 1 ;
- ☐ 1 : compteur = 1 ; 5 : compteur = 2 ; 5.5 compteur = 1 ;
- ☐ 1 : compteur = 1 ; 5 : compteur = 2 ; 5.5 compteur = 3 ;
- ☐ 1 : compteur = 1 ; 5 : compteur = 1 ; 5.5 compteur = 2 ;
- ☐ Il y a une erreur à la compilation

Question 11 Qu'est-ce qu'une classe abstraite?

- ☐ C'est une classe qui a toutes ses méthodes virtuelles
- ☐ C'est une classe qui n'a aucune donnée membre
- ☐ C'est une classe qui possède une méthode virtuelle pure
- ☐ C'est l'équivalent d'une interface en JAVA



Question 19 Qu'affiche le programme suivant?

```
class LaBase{
public:
  template <int NB>
  virtual void affiche() {std::cout << "LaBase" << NB << std::endl;}
};

class LaDerivee : public LaBase{
public:
  template <int NB>
  virtual void affiche() {std::cout << "LaDerivee" << NB << std::endl;}
};

int main(){
   LaDerivee  dede;
   LaBase & bibi = dede;
   bibi.affiche<5>();
   return 0;
}
```

- ☐ LaBase 5
- ☐ Il y a une erreur à la compilation
- ☐ LaDerivee 5

Question 20 Qu'affiche le programme suivant?

```
class LaBase{
public:
  void affiche() {std::cout << "LaBase" << std::endl;}
};

class LaDerivee : public LaBase{
public:
  void affiche() {std::cout << "LaDerivee" << std::endl;}
};

int main(){
   LaDerivee  dede;
   LaBase & bibi = dede;
   bibi.affiche();
   return 0;
}
```

- ☐ Il y a une erreur à la compilation
- ☐ LaDerivee
- ☐ LaBase

Question 20 Comment mettre en œuvre une liste de type en C++, sans recourir à des fonctionnalités du compilateur C++11? Donner un métaprogramme qui donne la longueur d'une liste de types.

Question 17 ♣ On considère l'exécution du petit programme suivant:

```
class Pixel{
 public :
   virtual void affiche();
};

class PixelNB : public Pixel {
   int intensite;
 public :
   PixelNB(int i=0): intensite(i){}
   void affiche();
};

class Image {
  Pixel * p;
public :
  Image() {p=new PixelNB[5*5];}
  // ... Reste de la classe ...
};

int main(){
  Image im;
   // ... Reste du code ...
  return 0;
}
```

Sachant qu'un `int` occupe 4 octets sur la machine d'exécution, et qu'un pointeur occupe 8 octets, quelle est la taille mémoire occupée par `im` (y compris les ressources supplémentaires allouées pour `im`)?

- ☐ 108 octets
- ☐ 208 octets
- ☐ 8 octets
- ☐ 308 octets
- ☐ *Aucune de ces réponses n'est correcte.*

Question 18 ♣ Qu'est-ce que le type statique d'une variable et plus généralement d'un objet?

- ☐ Pour un pointeur p, il s'agit du type pointeur sur le type avec lequel a été défini l'objet pointé par p
- ☐ Il s'agit du type d'un objet alloué dans le tas avec `new`
- ☐ Il s'agit des fonctions de la portée globale qui sont internes à un module, et avec lesquelles aucune édition de lien n'est possible
- ☐ Il s'agit du type correspondant à la déclaration d'une variable
- ☐ Les données membres d'une classe (mot clé `static`) sont de type statique
- ☐ *Aucune de ces réponses n'est correcte.*

Question 14 Que fait le programme suivant?

```
class Mystere{
   int d;
public:
  operator int() const {return d+5;}
  Mystere(int i=10) : d(i+1){}
  int operator()(int i) const {return d+i;}
};

int main(){
   std::cout << Mystere(8) << " puis " <<  Mystere(9)(3) << std::endl;
   return 0;
}
```

- ☐ 19 puis 15
- ☐ 9 puis 15
- ☐ 19 puis 13
- ☐ 14 puis 13

Question 15 ♣ Qu'est-ce que le type dynamique d'une variable et plus généralement d'un objet?

- ☐ Pour une référence, il s'agit du type correspondant à celui de l'objet référencé, au moment de l'exécution d'un programme
- ☐ Il s'agit du type d'un objet qui peut changer de taille au cours de l'exécution d'un programme (par exemple un `vector`)
- ☐ Il s'agit du type d'un objet alloué dans le tas avec `new`
- ☐ Il s'agit du type des objets qui ne sont pas des variables
- ☐ *Aucune de ces réponses n'est correcte.*

Question 16 Qu'affiche le programme suivant?

```
#include <iostream>
template <int T>
struct X{
    enum {res = T+X<T-1>::res};
};

template <>
struct X<1>{
    enum {res = 1};
};

int main() { std::cout << X<5>::res << std::endl; return 0;}
```

- ☐ 15
- ☐ 9
- ☐ Il y a une erreur à la compilation

Question 12 Qu'affiche le programme suivant?

```
class LaBase{
public:
  virtual void affiche() {std::cout << "LaBase" << std::endl;}
};

class LaDerivee : public LaBase{
public:
  virtual void affiche(int i=5) {std::cout << "LaDerivee" << std::endl;}
};

int main(){
   LaDerivee  dede;
   LaBase & bibi = dede;
   bibi.affiche(8);
   return 0;
}
```

- ☐ LaBase
- ☐ Il y a une erreur à la compilation
- ☐ LaDerivee

Question 13 ♣ Quelles affirmations sont justes, concernant les *smart-pointers* offerts par les dernières normalisations du C++.

- ☐ La fin de vie d'un objet pointé par un seul `weak_ptr` au cours d'un programme correspond à la fin de vie de ce `weak_ptr`
- ☐ La fin de vie d'un objet pointé par un seul `unique_ptr` au cours d'un programme correspond à la fin de vie de ce `unique_ptr`
- ☐ La fin de vie d'un objet pointé par un seul `shared_ptr` au cours d'un programme correspond à la fin de vie de ce `shared_ptr`
- ☐ Un objet peut simultanément être pointé par plusieurs `weak_ptr`
- ☐ Un objet peut simultanément être pointé par plusieurs `shared_ptr`
- ☐ Un objet peut simultanément être pointé par plusieurs `unique_ptr`
- ☐ Les *smarts-pointers* sont une abstraction des pointeurs intégrant la notion de propriété
- ☐ *Aucune de ces réponses n'est correcte.*