Trimestre Hiver, 2015

Mohamed Lokbani

IFT1169 – Examen Intra –

Inscrivez tout de suite : votre nom et le code permanent.

Nom : _____| Prénom(s) : _____|

Signature : _____| Login : _____|

Date : mardi 24 février 2015

Durée : 2 heures (de 18h30 à 20h30)

Local : Z-310, Pavillon Claire McNicoll

Directives :

- Toute documentation est permise.
- Calculatrice **non** permise.

- Répondre directement sur le questionnaire.
- Les réponses **doivent être brèves, précises, claires et nettement présentées.**

1._____/20 (1.1 à 1.10)

2._____/22 (2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7)

3._____/38 (3.1, 3.2)

4._____/20 (4.1)

Total : _____/100

Directives officielles

* Interdiction de toute communication verbale pendant l'examen.

* Interdiction de quitter la salle pendant la première heure.

* L'étudiant qui doit s'absenter après la première heure remettra sa carte d'étudiant au surveillant, l'absence ne devant pas dépasser 5 minutes. Un seul étudiant à la fois peut quitter la salle.

* Toute infraction relative à une fraude, un plagiat ou un copiage est signalée par le surveillant au directeur de département ou au professeur qui suspend l'évaluation.

F.A.S

Exercice 1 (20 points) Répondez par « vrai » ou « faux » en y incluant une très courte explication.

1.1 [VRAI | FAUX] Le titre du cours IFT1169 est: « Programmation mobile à plateforme libre ».

1.2 [VRAI | FAUX] Le langage C++ a été normalisé pour la première fois en 2003?

1.3 [VRAI | FAUX] Le langage C++ a récupéré la notion de référence du langage C.

1.4 [VRAI | FAUX] Le langage C++ n'impose pas l'encapsulation des membres dans leurs classes.

1.5 [VRAI | FAUX] Le nom d'une méthode définie par une classe doit nécessairement être précédé du nom de la classe, suivi immédiatement des caractères « :: ».

1.6 [VRAI | FAUX] Le mécanisme de la surcharge est un polymorphisme statique.

1.7 [VRAI | FAUX] Il est conseillé d'utiliser le mot clé « virtual » devant la déclaration du destructeur de la classe de base.

1.8 [VRAI | FAUX] Il n'est pas possible d'imbriquer un espace de noms dans un autre espace de noms.

1.9 [VRAI | FAUX] Si une fonction ne lance aucune exception, on peut la déclarer avec une clause throw vide.

1.10 [VRAI | FAUX] L'héritage multiple est facile à utiliser lorsque les superclasses n'ont rien en commun.

Exercice 2 (22 points)

```
 1  class A {
 2  protected:
 3        void f(){cout <<"A:f\n";};
 4   public:
 5        void g(){cout <<"A:g\n";};
 6  };
 7  class B: public A {
 8  public:
 9        using A::f;
10  };
11  class C: public A {
12  public:
13        void g(int){cout <<"C:g\n";};
14  };
15  class D: public A {
16  public:
17        void g(int){cout <<"D:g\n";};
18        using A::g;
19  };
```

2.1 (2 points) Expliquez l'impact de la déclaration de la ligne « **13** ».

2.2 (2 points) Expliquez l'impact de la déclaration de la ligne « **17** ».

2.3 (2 points) Expliquez l'impact de la déclaration de la ligne « **18** ».

2.4 (4 points) Tout en expliquant brièvement votre réponse, l'appel de la ligne ci-dessus est-il possible? Si c'est le cas, donnez l'affichage en sortie.

```
B b;
b.f(); // cet appel est-il possible?
```

2.5 (4 points) Tout en expliquant brièvement votre réponse, l'appel de la ligne ci-dessus est-il possible? Si c'est le cas, donnez l'affichage en sortie.

```
A a;
a.f(); // cet appel est-il possible?
```

2.6 (4 points) Tout en expliquant brièvement votre réponse, l'appel de la ligne ci-dessus est-il possible? Si c'est le cas, donnez l'affichage en sortie.

```
D d;
d.g(); // cet appel est-il possible?
```

2.7 (4 points) Tout en expliquant brièvement votre réponse, l'appel de la ligne ci-dessus est-il possible? Si c'est le cas, donnez l'affichage en sortie.

```
C c;
c.g(); // cet appel est-il possible?
```

Exercice 3 (38 points)

À noter que la déclaration et la définition des classes « A », « B », « C », « D » et « E » ne contiennent aucune erreur de syntaxe.

```
 1      class A {
 2      private:
 3            void pri();
 4            A(int a);
 5      protected:
 6            virtual void pro() {}
 7      public:
 8            void pub1() { cout<<"A.pub1()\n"; }
 9            virtual void pub2() { cout<<"A.pub2()\n"; }
10            virtual void pub3() { cout<<"A.pub3()\n"; }
11            virtual void pub4() { cout<<"A.pub4()\n"; }
12            virtual void pub5() { cout<<"A.pub5()\n"; }
13            virtual void pub6() { cout<<"A.pub6()\n"; }
14            virtual void pub7() { cout<<"A.pub7()\n"; }
15            virtual void pub8() { cout<<"A.pub8()\n"; }
16            void pub9() { cout<<"A.pub9()\n"; }
17            virtual void pub10() { cout<<"A.pub10()\n"; }
18            void pub11() { cout<<"A.pub11()\n"; }
19            explicit A() {}
20            virtual ~A() {}
21     };
22     class B : public A {
23     private:
24            void pri() { cout<<"B.pri()\n"; }
25     protected:
26            virtual void pub4() { cout<<"B.pub4()\n"; }
27            void pub6() { cout<<"B.pub6()\n"; }
28     public:
29            void pro() { cout<<"B.pro() "; B::pri(); }
30            void pub1() { cout<<"B.pub1()\n"; }
31            void pub2() { cout<<"B.pub2()\n"; }
32            void pub5() { cout<<"B.pub5()\n"; }
33            virtual void pub7() { cout<<"B.pub7()\n"; }
34            virtual void pub8() { cout<<"B.pub8()\n"; }
35            virtual void pub9() { cout<<"B.pub9()\n"; }
36            void pub10() { cout<<"B.pub10()\n"; }
37            void pub11() { cout<<"B.pub11()\n"; }
38            explicit B() {}
39     };
40     class C : protected B {
41     public:
42            void pub4_() { cout<<"C.pub4_() "; B::pub4(); }
43            virtual void pub5() { cout<<"C.pub5()\n"; }
44     };
45     class D : private B {
46     public:
47            void pub4_() { cout<<"D.pub4_() "; B::pub4(); }
48     };
49     class E : public B {
50     public:
51            virtual void pub4_() { cout<<"E.pub4_()\n"; }
52            virtual void pub7() { cout<<"E.pub7()\n"; }
53            virtual void pub8() { cout<<"E.pub8()\n"; }
54            virtual void pub9() { cout<<"E.pub9()\n"; }
55            virtual void pub10() { cout<<"E.pub10()\n"; }
56            virtual void pub11() { cout<<"E.pub11()\n"; }
57     };
```

Le tableau ci-dessous contient 55 instructions ordonnées. Ces instructions utilisent les classes « A », « B », « C », « D » et « E » définies précédemment.

Le tableau est composé de 4 colonnes :

- La 1ere colonne représente, le numéro de l'instruction.
- La 2e colonne contient l'instruction en question.
- La 3e colonne contient l'état de l'instruction à la compilation. Si elle compile sans erreur, on met la lettre « C ». Par contre, si elle provoque une erreur de compilation, on met la lettre « E ».
- La 4e colonne est en rapport avec l'affichage obtenu suite à l'exécution de l'instruction. Il y affichage si bien sûr l'instruction compile avec succès et si elle génère un affichage en sortie.

Sur les 55 instructions, 7 (ni plus ni moins) vont provoquer une erreur de compilation. De ce fait, ces 7 instructions ne vont pas générer un affichage en sortie. Ces 7 instructions n'ont pas d'incidence sur les instructions qui compilent correctement. Pour garantir cela, nous allons mettre en commentaire ces 7 instructions. Le programme va donc compiler et s'exécuter correctement par la suite.

3.1 Commencez par repérer les 7 instructions problématiques en y mentionnant la lettre « E » sur la ligne correspondante dans le tableau ci-dessous. Pour chaque instruction, expliquez brièvement l'erreur en question (un espace est réservé dans ce sens à la page -8-). (7 points)

3.2 Par la suite, complétez les 3e et 4e colonnes pour les instructions qui compilent et qui génèrent un affichage. Les explications pour cette étape ne sont pas nécessaires. (31 points).

No	Instruction	C/E	Affichage (s'il y a lieu)
1	`A* ap=new B();`		
2	`ap->pub1();`		
3	`(new B()).pub1();`		
4	`ap->pub2();`		
5	`(new B()).pub2();`		
6	`B b;`		
7	`b.A::pub1();`		
8	`b.pro();`		
9	`B* bp=new B;`		
10	`bp->pub3();`		
11	`C c;`		
12	`c.pub3();`		
13	`c.pub4();`		
14	`c.pub4_();`		
15	`c.pub5();`		
16	`D d;`		
17	`d.pub3();`		
18	`d.pub4();`		
19	`d.pub4_();`		
20	`E e;`		
21	`e.pub4();`		
22	`delete ap;`		

23	`ap = new E();`		
24	`ap->pub4();`		
25	`ap->pub5();`		
26	`ap->pub6();`		
27	`ap->pub7();`		
28	`delete bp;`		
29	`bp = new E();`		
30	`e.pub8();`		
31	`e.A::pub8();`		
32	`e.B::A::pub8();`		
33	`e.B::pub8();`		
34	`ap->pub8();`		
35	`bp->pub8();`		
36	`e.pub9();`		
37	`e.A::pub9();`		
38	`e.B::A::pub9();`		
39	`e.B::pub9();`		
40	`ap->pub9();`		
41	`bp->pub9();`		
42	`e.pub10();`		
43	`e.A::pub10();`		
44	`e.B::A::pub10();`		
45	`e.B::pub10();`		
46	`ap->pub10();`		
47	`bp->pub10();`		
48	`e.pub11();`		
49	`e.A::pub11();`		
50	`e.B::A::pub11();`		
51	`e.B::pub11();`		
52	`ap->pub11();`		
53	`bp->pub11();`		
54	`delete ap;`		
55	`delete bp;`		

Erreur no 1, instruction numéro _____, explications :

Erreur no 2, instruction numéro _____, explications :

Erreur no 3, instruction numéro _____, explications :

Erreur no 4, instruction numéro _____, explications :

Erreur no 5, instruction numéro _____, explications :

Erreur no 6, instruction numéro _____, explications :

Erreur no 7, instruction numéro _____, explications :

Exercice 4 (20 points) On veut éliminer les lettres répétitives dans une phrase donnée. Par exemple pour la phrase « Ceci est un exemple », le résultat de cette démarche est « Ceci stunxmpl ».

Compléter la fonction « `sans_repetitions(string&);` » du programme ci-dessous. Cette fonction ne retourne rien et n'accepte qu'un seul argument passé par référence du type « string ». L'utilisation des fonctionnalités de la « STL » n'est pas permise.

```
1   int main(){
2         string a_string;
3         cout << "Tapez une phrase: ";
4         getline(cin,a_string);
5         sans_repetitions(a_string);
6         cout << "\nLe resultat est: " << a_string << "\n";
7         return 0;
8   }
```

```
>./exo4
Tapez une phrase: Ceci est un exemple
Le resultat est: Ceci stunxmpl
```

Lors de la correction, je vais tenir compte de la clarté de votre code et de son optimisation.