Trimestre Automne, 2017

Mohamed Lokbani

IFT1169 – Examen Final –

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

Nom : _____| Prénom(s) : _____|

Signature : _____| Login : _____|

Date : Vendredi 15 décembre 2017

Durée : 3 heures (de 16h30 à 19h30)

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

2._____/15

3._____/20

4._____/15

5._____/10

6._____/20

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 compilateur gcc permet la génération d'information de débogage à l'aide de l'option "-x".

1.2 [VRAI | FAUX] Si la classe de base a un constructeur par défaut, aucun constructeur de la classe dérivée n'a besoin d'appeler explicitement le constructeur de la classe de base.

1.3 [VRAI | FAUX] Si la classe de base a un constructeur avec arguments, mais pas de constructeur par défaut, on ne peut pas créer une instance de la classe dérivée sauf si le constructeur de la classe dérivée appelle explicitement un des constructeurs de la classe de base.

1.4 [VRAI | FAUX] Si la classe de base n'a aucun constructeur, on ne peut pas créer une instance de la classe dérivée.

1.5 [VRAI | FAUX] En C++, les méthodes d'une classe donnée sont polymorphiques par défaut.

1.6 [VRAI | FAUX] Les méthodes statiques ne peuvent pas être polymorphiques.

1.7 [VRAI | FAUX] Une classe dérivée ne peut pas avoir une méthode ayant le même nom que sa classe de base.

1.8 [VRAI | FAUX] Dans le fragment de code suivant, sachant que le destructeur de la classe de base est virtuel, le résultat de l'instruction -8- fait en sorte qu'on exécute d'abord le destructeur de la classe de base, par la suite on exécute celui de la classe dérivée.

```
baseClasse *a = new deriveeClasse;
delete (a); // l'instruction -8-
```

1.9 [VRAI | FAUX] Le fragment de code suivant affiche en sortie « `0:0:1:0` » si on fournit en entrée la lettre « G ».

```
int Valeur;
cin >> Valeur; // G en entrée
cout << cin.eof() << ":" << cin.fail() << ":" << cin.bad() << ":" << cin.good();
```

1.10 [VRAI | FAUX] Vu qu'on ne flushe jamais la sortie dans ce fragment de code, il ne va rien afficher en sortie.

```
int main(int argc, char** argv) {
    cout << "Ceci est un test!";
    return 0;
}
```

Exercice 2 (15 points) Soit le programme suivant :

```
 1  #include <iostream>
 2  #include <exception>
 3
 4  void g(){
 5      throw "Exception";
 6  }
 7
 8  void f(){
 9      int* pI = new int(0);
10      g();
11      delete pI;
12  }
13
14  int main(){
15      f();
16      return 0;
17  }
```

2.1 Donner le déroulement des instructions pour le précédent programme.

- Le programme commence par exécuter la fonction « main » à la ligne -14-.
- Par la suite, il y a un appel à la fonction « f() » à la ligne -15-.

À vous la suite ...

2.2 Vous êtes arrivés à la conclusion que le précédent programme ne va pas s'exécuter jusqu'au bout. Expliquer les raisons.

2.3 Comment peut-on corriger un tel problème tout en gardant le principe de fonctionnement du programme. Vous n'êtes autorisés qu'à ajouter des instructions.

Exercice 3 (20 points) ce programme compile et s'exécute correctement.

```
 1
 2  #include <iostream>
 3
 4  using namespace std;
 5
 6  class A {
 7  public:
 8      A() { cout << "A cteur" << endl; }
 9      A(const A& a) { cout << "A copie cteur" << endl; }
10      virtual ~A() { cout << "A dteur" << endl; }
11      virtual void foo() { cout << "A foo()" << endl; }
12      virtual A& operator=(const A& rhs) { cout << "A op=" << endl; return *this; }
13  };
14
15  class B : public A {
16  public:
17      B() { cout << "B cteur" << endl; }
18      virtual ~B() { cout << "B dteur" << endl; }
19      virtual void foo() { cout << "B foo()" << endl; }
20  protected:
21      A uneInstanceA;
22  };
23
24  A foo(A& entree) {
25      entree.foo();
26      return entree;
27  }
28
29  int main() {
30      B unB;
31      B unAutreB;
32      A unA;
33      unAutreB = unB;
34      unA = foo(unAutreB);
35
36      return 0;
37  }
```

Que va afficher le précédent programme? Expliquez brièvement votre démarche.

Exercice 4 (15 points) Écrire la fonction générique « log » qui accepte un nombre variable d'arguments permettant de réaliser les différents appels dans la fonction « main » ci-dessous :

```
1   int main() {
2
3       log(1,4.3, "Bonjour");
4       log('a', "test", 78L, 5);
5       log(3);
6
7       return 0;
8   }
```

Affichage en sortie :

```
1   >./Exo4
2   1 , 4.3 , Bonjour ,
3   a , test , 78 , 5 ,
4   3 ,
5   >Exit code: 0
```

Exercice 5 (10 points) Tout en développant brièvement votre démarche, dessiner le résultat obtenu par le fragment de code suivant :

```
 1  bool Exo5::OnInit() {
 2      const wxString Titre("Exo5");
 3      TexteFrame *frame = new TexteFrame(Titre);
 4      frame->Show(TRUE);
 5      SetTopWindow(frame);
 6
 7      return true;
 8  }
 9  TexteFrame::TexteFrame
10              (const wxString titre)
11              : wxFrame
12                 ( (wxFrame *) NULL,
13                    -1,
14                    titre
15                 ){
16         wxPanel *UnPanneau = new wxPanel(this, -1);
17         wxBoxSizer *intermed = new wxBoxSizer(wxVERTICAL);
18         wxGridSizer *UneGrille = new wxGridSizer(4, 6, 5, 5);
19         for (int i=0; i<24; i++){
20                int style = ((i+(i/6))%2)==0 ? wxALIGN_CENTER : wxEXPAND;
21                wxPanel *element = new wxPanel(UnPanneau, -1);
22                element->SetBackgroundColour(*wxBLUE);
23                UneGrille->Add(element, 0, style);
24         }
25         intermed->Add(UneGrille, 1, wxALL | wxEXPAND, 5);
26         UnPanneau->SetSizer(intermed);
27  }
28  TexteFrame::~TexteFrame(){}
```

Exercice 6 (20 points) Tout en utilisant au maximum les notions apprises dans les STL (conteneurs, algorithmes, méthodes, etc.) écrire un programme en C++ qui permet de valider qu'une expression a un nombre correctement balancé de signes de ponctuation et dans le bon ordre. Les signes de ponctuation visés sont : `[ ]` (crochets), `{ }` (accolades) et `( )` (parenthèses).

Ainsi,

L'expression `(4+{8-[22+8]*})` contient un nombre valide et dans le bon ordre (ouvert/fermé) de signes de ponctuation.

L'expression `({5+8])` contient un nombre invalide de signes de ponctuation.

```
1  int main(){
2    cout<<testExpression("([(2+11)]+)")<<endl;
3    cout<<testExpression("(4+{8-[22+8]*}]")<< endl;
4    cout<<testExpression("({5+8])")<< endl;
5
6    return 0;
7  }
```

Affichage en sortie :

```
1  >./Exo6
2  1
3  0
4  0
5  >Exit code: 0
```

Joyeuses fêtes et bonne année!