Trimestre Automne, 2024

Mohamed Lokbani

IFT1169 – Examen Intra –

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

Nom : _____| Prénom(s) : _____|

Signature : _____| Login : _____|

Date : mercredi 16 octobre 2024

Durée : 2 heures (de 18h30 à 20h30)

Local : Z-330, Pavillon Claire McNicoll

Directives :

- Toute documentation est permise.
- Appareils électroniques **non** permis.

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

1._____/20 (1.1 à 1.10)

2._____/15 (2.1 à 2.5)

3._____/25 (3.1 à 3.8)

4._____/20 (4.1)

5._____/20 (5.1)

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] Un Makefile est utilisé pour automatiser le processus de compilation.

1.2 [VRAI | FAUX] La commande « make » exécute les instructions du « Makefile ».

1.3 [VRAI | FAUX] Un « Makefile » ne peut contenir qu'une seule règle.

1.4 [VRAI | FAUX] Les dépendances dans un « Makefile » sont spécifiées après le caractère « **:** ».

1.5 [VRAI | FAUX] Les commandes dans un « Makefile » doivent commencer par un espace.

1.6 [VRAI | FAUX] Les commentaires dans un « Makefile » commencent par le caractère « **#** ».

1.7 [VRAI | FAUX] Un « Makefile » ne peut pas inclure d'autres « Makefiles ».

1.8 [VRAI | FAUX] Les règles dans un « Makefile » peuvent être conditionnelles.

1.9 [VRAI | FAUX] Les variables dans un « Makefile » peuvent être substituées avec la syntaxe « **$(VARIABLE)** ».

1.10 [VRAI | FAUX] Les « Makefiles » sont spécifiques au compilateur utilisé.

Exercice 2 (15 points) soit les deux paires de fichiers « **A.h/A.cpp »** et « **B.h/B.cpp »** associés respectivement aux classes « **A** » et « **B** ». Soit la méthode « **alpha** » une des méthodes de la classe « **A** » et « **beta** » une des méthodes de la classe « **B** ». La méthode « **alpha** » de la classe « **A** » fait appel à la méthode « **beta** » de la classe « **B** ». La méthode « **main** » est définie dans un fichier à part, « **main.cpp** ».

2.1 Quelle(s) information(s) relative(s) à la classe « **B** » doit-on ajouter dans la paire de fichiers associée à la classe « **A** » pour que cette paire compile correctement?

2.2 Quel est le type du fichier créé suite à la compilation des fichiers de la paire « **A** »?

2.3 Est-il possible de produire un exécutable en utilisant uniquement les fichiers de la paire « **A** »?

2.4 Si vous commettez une erreur typographique et faite en sorte que la méthode « **alpha** » appelle la méthode « **beto** » au lieu de la méthode « **beta** », quelle est l'erreur générée lors de la compilation de la paire de fichiers de la classe « **A** »?

2.5 Écrire les instructions relatives au compilateur « **g++** », permettant de réaliser l'édition de liens des trois fichiers « **main.cpp** », « **A.cpp** » et « **B.cpp** », et qui génère en sortie l'exécutable « **exo2.exe** ».

Exercice 3 (25 points) soit le fragment de code suivant qui est syntaxiquement correct.

```
 1 class Oiseau {
 2 public:
 3       virtual void Parle()=0;
 4       virtual void MangePoissons() {cout << "cela dépend" << endl;}
 5 };
 6 class Pigeon : public Oiseau {
 7 public:
 8       virtual void Parle() {cout << "Coo, Coo" << endl;}
 9 };
10 class Aigle : public Oiseau {
11 public:
12       virtual void Parle() {cout << "Squawk, Squawk" << endl;}
13       virtual void MangePoissons() {cout << "J'adore le poisson!" << endl;}
14       void MangePigeons() {cout << "Un succulent morceau!" << endl;}
15 };
16 class TeteBlancheAigle : public Aigle {
17 public:
18       virtual void Parle() {cout << "Je suis un spécimen en danger!" << endl;}
19 };
20 // Fonctions non membres:
21 void AigleParle(Aigle aig) { aig.Parle();}
22 void MangeurPoissons(Oiseau *oies) { oies->MangePoissons();}
```

Le corps de la méthode « **main** » est constitué d'un bloc à la fois des instructions des questions **3.1** à **3.8**. Les blocs sont indépendants l'un de l'autre. Indiquez pour chaque bloc s'il est correct ou non, en entourant la bonne réponse, et expliquez pourquoi. Si le bloc est incorrect, corrigez l'erreur. Un bloc est dit incorrect si au moins une des instructions qui le composent est incorrecte. Part incorrecte, nous entendons qu'une des instructions du bloc en question génère une erreur lors de la compilation et/ou de l'exécution du programme. Pour les blocs corrects, indiquez l'affichage obtenu en sortie.

3.1

`Oiseau *b0 = new Oiseau;` `b0->MangePoissons();`	Correct / Incorrect

Pourquoi?

3.2

`Oiseau *b1 = new Aigle;` `b1→MangePoissons();`	Correct / Incorrect

Pourquoi?

3.3

`Oiseau *b1 = new Aigle;` `b1→MangePigeons();`	Correct / Incorrect

Pourquoi?

3.4

`Aigle e1;`
`TeteBlancheAigle be1;`
`be1 = e1;`
	Correct / Incorrect

Pourquoi?

3.5

`TeteBlancheAigle be2;` `AigleParle(be2);`	Correct / Incorrect

Pourquoi?

3.6

`TeteBlancheAigle be3;` `be3.MangePigeons();`	Correct / Incorrect

Pourquoi?

3.7

`TeteBlancheAigle be4;` `MangeurPoissons(&be4);`	Correct / Incorrect

Pourquoi?

3.8

`TeteBlancheAigle *be5 = new TeteBlancheAigle;`
`Pigeon *p1;`
`p1 = be5;`
	Correct / Incorrect

Pourquoi?

Exercice 4 (20 points) ce programme compile et s'exécute correctement.

```
 1 #include <iostream>
 2 #include <fstream>
 3
 4 using namespace std;
 5
 6 void transfert(ifstream& entree,ofstream& sortie){
 7        char ch;
 8        bool flag = false;
 9        int n = 0;
10        while (entree.get(ch)) {
11               if (ch == 'l' && !flag) {
12                      flag = true;
13                      sortie << ch;
14                      n++;
15               } else if (ch != 'l') {
16                      flag = false;
17                      sortie << ch;
18               }
19        }
20        cout << "n: " << n << endl;
21 }
22
23 int main() {
24
25        ifstream ficentree("entree.txt");
26        ofstream ficsortie("sortie.txt");
27
28        transfert(ficentree,ficsortie);
29        return 0;
30 }
```

Le fichier « entree.txt » contient ce qui suit :

```
Dans une vieille bibliothèque poussiéreuse, une vieille femme feuilletait un livre
ancien. Les étagères étaient remplies de volumes reliés en cuir, chacun contenant
des histoires fascinantes. Elle tomba sur un manuscrit mystérieux, rempli
d'illustrations et de symboles énigmatiques. Curieuse, elle décida de l'emporter
chez elle pour le déchiffrer tranquillement, espérant découvrir des secrets oubliés
depuis longtemps.
```

Expliquer brièvement le fonctionnement du programme. Que va-t-il afficher en sortie ? Donner le contenu du fichier « sortie.txt » après l'exécution du programme.

Exercice 5 (20 points) la fonction « main » suivante est syntaxiquement correcte.

```
 1 int main() {
 2     Point3D p1(1, 2, 3);
 3     Point3D p2(4, 5, 6);
 4
 5     Point3D p3 = p1 * 2.5;
 6     std::cout << "p1 * 2.5 = " << p3 << std::endl;
 7
 8     std::cout << "Composante y de p1: " << p1(1) << std::endl;
 9
10     Point3D p4 = -p1;
11     std::cout << "-p1 = " << p4 << std::endl;
12
13     Point3D p5 = p1 + p2;
14     std::cout << "p1 + p2 = " << p5 << std::endl;
15
16     p1[0] = 10;
17     std::cout << "Nouvelle composante x de p1: " << p1 << std::endl;
18
19     bool sontEgaux = (p1 == p2);
20     std::cout << "p1 et p2 sont " << (sontEgaux ? "égaux" : "différents") << std::endl;
21
22     return 0;
23 }
```

Le squelette de la classe « **Point3D** » est comme suit :

```
class Point3D {
private:
    double x, y, z;

public:
// les diverses méthodes …
};
```

L'affichage obtenu en sortie :

```
p1 * 2.5 = (2.5, 5, 7.5)
Composante y de p1: 2
-p1 = (-1, -2, -3)
p1 + p2 = (5, 7, 9)
Nouvelle composante x de p1: (10, 2, 3)
p1 et p2 sont différents
```

Écrire uniquement le code des méthodes de la classe « **Point3D** » nécessaires au bon fonctionnement de la fonction « **main** » décrite précédemment. Il n'est donc pas nécessaire de surcharger votre code avec des méthodes « superflues ».