Trimestre Automne, 2023

Mohamed Lokbani

IFT1169 – Examen Intra –

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

Nom : _____| Prénom(s) : _____|

Signature : _____| Login : _____|

Date : jeudi 26 octobre 2023

Durée : 2 heures (de 16h30 à 18h30)

Local : Z-205, 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._____/20 (2.1)

3._____/15 (3.1, 3.2)

4._____/15 (4.1, 4.2)

5._____/30 (5.1 à 5.8)

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] Pour cette session d'automne 2023, le cours IFT1169 se donne avec la version 13.x du compilateur g++.

1.2 [VRAI | FAUX] On ne peut pas produire un programme exécutable (en C++) sans l'utilisation d'une fonction `main` dans le programme.

1.3 [VRAI | FAUX] La signature de la fonction `main` est : `int main(int x, char** y)`.

1.4 [VRAI | FAUX] Pour inclure dans votre programme le fichier d'entête `xyz.h`, créé par vous, vous allez utiliser l'instruction `#include <xyz.h>`.

1.5 [VRAI | FAUX] La commande `g++ -c toto.exe toto.cpp` permet de produire le programme exécutable `toto.exe`.

1.6 [VRAI | FAUX] Le programme utilise la fonction `sinus`. Nous devons fournir le calcul de la fonction à l'étape de compilation de mon programme.

1.7 [VRAI | FAUX] L'utilisation d'un compilateur est préférée à celle d'un préprocesseur.

1.8 [VRAI | FAUX] Dans un programme, on peut se passer de la directive `using namepsace std;`

1.9 [VRAI | FAUX] Lors de la compilation, il suffit de spécifier les fichiers sources. Les fichiers d'en-tête seront automatiquement inclus.

1.10 [VRAI | FAUX] L'option `-pedantic` permet de spécifier la version du standard *C++* à utiliser lors de la compilation d'un programme.

Exercice 2 (20 points) soit les 5 fichiers suivants, sans erreurs de syntaxes, écrits dans un langage *C++* :

```
/**********************
* Fichier: Alpha.h *
**********************/
#ifndef Alpha_H
#define Alpha_H
class Alpha {
// etc.
};
#endif
```

```
/*********************
* Fichier: Beta.h *
*********************/
#ifndef Beta_H
#define Beta_H
#include "Alpha.h"
class Beta : public Alpha {
// etc.
};
#endif
```

```
/************************
* Fichier: Alpha.cpp *
************************/
#include "Alpha.h"
// définitions des méthodes
// de la classe Alpha
```

```
/***********************
* Fichier: Beta.cpp *
***********************/
#include "Beta.h"
// définitions des méthodes
// de la classe Beta
```

```
/*********************
* Fichier: Charlie.cpp *
*********************/
#include "Alpha.h"
#include "Beta.h"
int main() {
//etc.
}
```

2.1 Nous avons reçu le makefile ci-dessous. En tentant de l'exécuter, nous avons obtenu une panoplie d'erreurs à l'écran. On vous demande de trouver ces erreurs, de les expliquer puis de les corriger. S'il manque des éléments dans le fichier makefile, ajoutez-les, tout en expliquant brièvement pourquoi il serait nécessaire de les inclure. À noter que la cible globale du makefile est all.

```
1  # Fichier makefile de départ
2  Beta.o: Beta.cpp Alpha.cpp Beta.h
3  gcc -Wall -g -c Beta.h
4  Alpha.o: Alpha.cpp Alpha.h
5  gcc -Wall -g -c Alpha.h
6  Charlie.o: Charlie.cpp Alpha.cpp Beta.cpp
7  gcc -Wall -g -c Charlie.h
8  all: Charlie.o Alpha.o Beta.o
9  gcc -Wall -g -o test Charlie.o Alpha.cpp Beta.cpp
```

Exercice 3 (15 points) ce programme compile et s'exécute :

```
01  #include<iostream>
02  using namespace std;
03  class Alpha {
04  public:
05          Alpha(){ cout <<"9";}
06  };
07  class Bravo: virtual public Alpha {
08  public:
09          Bravo(){cout <<"6";}
10  };
11  class Charlie: virtual public Alpha {
12  public:
13          Charlie(){cout<<"4";}
14  };
15  class Delta:public Bravo,public Charlie {
16  public:
17          Delta(){cout<<"3";}
18          Delta(const Delta & obj){cout <<"1";}
19  };
20
21  int main() {
22          Delta d1;
23          Delta d(d1);
24          cout << endl;
25          return 0;
26  }
```

3.1 Tout en développant votre réponse, que va afficher en sortie le précédent programme?

3.2 On supprime maintenant `virtual` des deux lignes 7 et 11, comme suit :

```
class Bravo: public Alpha { …
class Charlie: public Alpha { …
```

Le programme compile et s'exécute correctement après ces modifications. Tout en développant votre réponse, que va-t-il afficher en sortie (après ces modifications)?

Exercice 4 (15 points) soit la classe générique `Test` :

```
01  #include <iostream>
02
03  using std::ostream;
04  using std::cout;
05
06
07  template<class T>
08  class Test{
09  public:
10          Test(T x): valeur(x) {}
11          Test operator+(Test &x);
12          friend ostream& operator<<<>(ostream&, const Test<T>&);
13  private:
14          T valeur;
15  };
16
17  int main(){
18          Test<int> t1(6);
19          cout << "t1: " << t1 ;
20          Test<int> t2(12);
21          cout << "t2: " << t2 ;
22          Test<int> t3 = t1 + t2;
23          cout << "t3: " << t3 ;
24
25          return 0;
26  }
```

4.1 Tout en expliquant votre démarche, compléter l'opérateur décrit à la ligne 11 :

```
Test operator+(Test &x);
```

La fonction `main` donne un exemple de son utilisation.

4.2 Tout en expliquant votre démarche, compléter l'opérateur de la ligne 12 :

```
friend ostream& operator<<<>(ostream&, const Test<T>&);
```

Voici l'affichage obtenu en sortie à la suite de l'exécution du programme de l'exercice 4 :

```
t1: 6
t2: 12
t3: 18
```

Exercice 5 (30 points) soit le fragment de code suivant :

```
/**********************
* Fichier: Vehicule.h *
**********************/

#include <iostream>

#ifndef H_VEHICULE
#define H_VEHICULE

class Vehicule {
public:
	Vehicule(int nbRoues = 0);
	virtual ~Vehicule();
	virtual void Affiche() const;
	int GetRoues() const;
	void SetRoues(int Roues );
private:
	int m_nbRoues;
};

#endif
```

```
/**********************
* Fichier: Vehicule.cpp *
**********************/

#include "Vehicule.h"


Vehicule::Vehicule(int Roues){
	m_nbRoues = Roues;
}
int Vehicule::GetRoues() const{
	return m_nbRoues;
}
void Vehicule::SetRoues(int Roues){
	m_nbRoues = Roues;
}
void Vehicule::Affiche() const{
	std::cout << "Vehicule";
}
Vehicule::~Vehicule() {}
```

```
/**********************
* Fichier: Moto.h *
**********************/
#include <iostream>
#include "Vehicule.h"

#ifndef H_MOTO
#define H_MOTO

class Moto:public Vehicule{
public:
	Moto(int Roues = 2);
	~Moto();
};

#endif
```

```
/**********************
* Fichier: Moto.cpp *
**********************/
#include "Moto.h"

// Constructeur de Moto à compléter
Moto::Moto (int Roues);

Moto::~Moto (){}
```

```
/**********************
* Fichier: Voiture.h *
**********************/
#include <iostream>
#include "Vehicule.h"

#ifndef H_VOITURE
#define H_VOITURE

//====================
// Classe Voiture
//====================
class Voiture:public Vehicule {
public:
	Voiture (int Roues = 4);
	void Affiche() const;
};

#endif
```

```
/**********************
* Fichier: Voiture.h *
**********************/
#include "Voiture.h"

// Constructeur de Voiture à compléter
Voiture::Voiture(int Roues);

void Voiture::Affiche() const{
	std::cout << "Voiture";
}
```

```
/***********************
* Fichier: Parc.h *
***********************/
#include <iostream>
#include "Vehicule.h"

//============================
// Classe Parc de stationnement
//============================

#ifndef H_PARC
#define H_PARC
class Parc {
public:
	Parc(int maxVehicules = 20);
	~Parc();
	int NbStationnes() const;
	void Garer(Vehicule& v );
	int TotalRoues() const;
	void Affiche() const;
	Parc(const Parc&);
	Parc& operator=(const Parc&);
private:
	int m_nbVehicules;
	Vehicule **m_Vehicules;
};
#endif
```

```
/***********************
* Fichier: Parc.cpp *
***********************/
#include "Parc.h"

// Constructeur de recopie
// et opérateur= sont déjà codés.

// Constructeur de Parc à compléter

// Destructeur de Parc à compléter

// Méthode Garer à compléter
// Elle stocke le véhicule (l'argument)
// dans le tableau m_Vehicules

// Méthode Affiche à compléter
// Elle affiche en sortie le type
// du véhicule et son nombre de roues
// Comme exemple de format d'affichage:
// Voiture:4

// Méthode ToTalRoues à compléter
// Elle retourne le nombre total
// de roues dans le parc de
// stationnement

int Parc::NbStationnes() const {
	return m_nbVehicules;
}
```

```
/***********************
* Fichier: Main.cpp *
***********************/
#include <iostream>
#include "Vehicule.h"
#include "Voiture.h"
#include "Moto.h"
#include "Parc.h"

int main(){
	Parc Parking;
	Voiture chevy;
	Voiture camry;
	Moto harley(3);
	Moto honda;
	Parking.Garer(chevy);
	Parking.Garer(honda);
	Parking.Garer(harley);
	Parking.Garer(camry);
	Parking.Affiche( );
	std::cout << std::endl;
	std::cout << "Le parc contient : " << Parking.TotalRoues() \
                                                << " Roues" << std::endl;
	return 0;
}
```

5.1 Citez au moins deux instructions du programme qui forcent l'utilisation de la ligature dynamique par ce dernier.

5.2 Le programme contient-il une classe abstraite ? Si oui, nommez-la.

5.3 Nommez les classes qui dérivent de la classe « `Vehicule` ».

5.4 La classe « `Parc` » peut-elle conceptuellement parlant devenir une classe de base pour les autres classes du programme ?

5.5 Le compilateur va laisser la classe « `Voiture` » redéfinir la méthode « `GetRoues` », mais ce n'est pas une très bonne idée de le faire ? Est-ce vrai ?

5.6 La méthode « `SetRoues` » ne peut être utilisée de manière polymorphique, car ce n'est pas une méthode virtuelle (ou virtuelle pure). Est-ce vrai ?

5.7 Écrivez le code des méthodes suivantes :

(a) Constructeur de la classe « Moto »

(b) Constructeur de la classe « Voiture »

(c) Constructeur de la classe « Parc »

(d) Destructeur de la classe « Parc »

(e) Méthode « Garer » de la classe « Parc »

(f) Méthode « Affiche » de la classe « Parc »

(g) Méthode « TotalRoue » de la classe « Parc »

5.8 Tout en justifiant votre réponse que va afficher en sortie le précédent programme ?