

Nom et prénom : Franchetti Thibaud

Travail écrit no. 1

(Durée : 2 périodes)

Directives :

- **ECRIVEZ VOS REPONSES DIRECTEMENT SUR LA DONNEE**
- Ne pas dégrafer le document
- Vous pouvez écrire au crayon
- Seule documentation autorisée : **la Quick Reference Card C++ (non annotée !)**

0.95

Question 1 (1.25 point)

Compléter la partie notée *<à compléter>* du programme ci-après, de telle sorte que celui-ci affiche à l'exécution :

```
01.1900
11.2018
01.2019
Exception : mois doit etre un entier entre 1 et 12
```

<à compléter>

```
int main() {
    try {
        cout << Date() << endl;
        // Date d1 = 1; // Sans les commentaires, serait refusé à la compilation
        Date d2(11, 2018);
        cout << d2++ << endl;
        cout << ++d2 << endl;
        Date d3(13, 2019);
    } catch (const MoisNonValide& e) {
        cout << e.what() << endl;
    }
    return EXIT_SUCCESS;
}
```

IMPORTANT

- Tout le code à produire est censé figurer dans le même fichier que *main*
 - Le code de *main* ne doit en aucun cas être modifié
 - Le code doit être écrit sans faire usage d'exceptions prédéfinies *donc on considère qu'il ne faut pas utiliser exception*
 - La classe *Date*
 - est supposée ne comporter que 2 données-membres : *mois* (1-12) et *annee*
 - ne doit comporter qu'une seule fonction *non* membre au plus
 - ne doit déclarer que les fonctions strictement nécessaires à l'exécution du *main* proposé, et rien d'autre¹ !
- ¹ cela signifie, en particulier, qu'aucun accesseur ne doit être proposé.
- Aucune fonction de la classe *Date* ne doit être codée en ligne (inline)
 - Appliquer l'encapsulation et éviter au maximum d'écrire du code redondant
 - Il n'est pas demandé de commenter les diverses fonctions de la classe *Date*

Votre réponse :

```
#include <iostream>
#include <cstdlib>
#include <iomanip>
#include <string>
using namespace std;
```

```
class MoisNonValide { // n'hérite pas de exception voir remarque p. 2
public:
    explicit MoisNonValide(const string& what_arg); { manque second constructeur
    const char* string what() const;
private:
    string what_arg;
};
```

```
MoisNonValide::MoisNonValide(const string& what_arg) : what_arg(what_arg) {}
string MoisNonValide::what() const {
    return what_arg;
}
```

```
class Date {
friend ostream& operator<< (ostream& lhs, const Date& rhs);
public:
    explicit Date(unsigned mois = 1, unsigned annee = 1900);
    Date& operator++();
    Date operator++(int);
private:
    unsigned mois;
    unsigned annee;
};
```


Votre réponse (suite) :

~~Date::Date(unsigned mois, unsigned annee) { mois/mois, annee/annee } }~~


```
ostream& operator<< (ostream& lhs, const Date& rhs) {  
    lhs << setfill('0') << setw(2) << mois << "."  
    << setfill('0') << setw(4) << annee;
```

return lhs;
}

↑ répétition inutile

~~Date::~~  ~~operator++()~~ {
 annee += (unsigned) (mois == 12);
 mois = (mois % 12) + 1;
 return *this; ✓

}

~~Date::~~  ~~operator++(int)~~ {
 Date tmp = *this;
 ++(*this); ~~inutile~~
 return tmp; ✓

}

```
Date::Date(unsigned mois, unsigned annee) {  
    if (mois == 0 || mois > 12) ✓ {  
        throw ArgumentNonValide; ✓ "Exception : mois doit être"  
        "en valeur entre 1 et 12";  
    }  
    this->mois = mois;  
    this->annee = annee; ✓
```

}

0.625

Question 2 (0.75 point)

Le code ci-après est censé afficher à l'exécution :

J = 0.5
c1 : i = 1
c2 : i = 2
c3 : i = 1
c3 : i = 2

```
1 #include <cstdlib>
2 #include <iostream>
3 using namespace std;
4
5 class C {
6     friend ostream& operator<<(ostream& os, const C& c);
7 public:
8     C(int i);
9     C(const C& c);
10    C& operator=(const C& c);
11    int getI() const;
12    static double getJ() const;
13 private:
14     const int i;
15     static const double J = 0.5;
16 };
17 static const double J = 0.5;
18 friend ostream& operator<<(ostream& os, const C& c) {
19     return os << "i = " << c.i;
20 }
21
22 C::C(int i) : i(i) {}
23
24 C::C(const C& c) : i(c.i) {}
25
26 C& C::operator=(const C& c) {
27     (int&) i = c.i; if (this != &c)
28     return *this;
29 }
30
31 int C::getI() const {return i;}
32 double C::getJ() const {return J;}
33
34 int main() {
35     cout << "J = " << C::getJ() << endl;
36     C c1 = 1;
37     C c2{2};
38     C c3(c1);
39     cout << "c1 : " << c1 << endl;
40     cout << "c2 : " << c2 << endl;
41     cout << "c3 : " << c3 << endl;
42     c3 = c2;
43     cout << "c3 : " << c3 << endl;
44     return EXIT_SUCCESS;
45 }
```

Ce code contient toutefois diverses erreurs ou maladresses.

Pour chacune d'entre elles, **indiquer le numéro de la ligne fautive et proposer un correctif C++**.

IMPORTANT

- La nature des 2 champs ("usuel constant" pour *i* et "static const" pour *J*) ne doit pas être modifiée
- Aucun champ ne doit être ajouté
- Aucune fonction amie ou fonction-membre ne doit être ajoutée
- Des points seront décomptés si des erreurs ou maladresses supplémentaires sont introduites

Votre réponse :

12. `static double getJ();` ✓
15. `static const double J;` ✓
17. ~~`static const double`~~ `J = 0.5;` ✓
18. `ostream& operator<< (ostream& os, const C& c) {` ✓
19. `return os << "i = " << c.i;` ✓
27. `if (this != &c) (int&) i = c.i;` *Pas faux... mais n'amène rien ici*
32. `double C::getJ() {return J;}` ✓
35. `cout << "J = " << C::getJ() << endl;` ✓
9. inutile, le compilateur fournit un constructeur de copie par défaut
24. inutile (cf. 9)

Votre réponse (suite) :

Question 3 (1.25 point)

a) (0.45 pts)

Soient les définitions de fonctions suivantes :

```
template <typename T, typename U> void f(T, U) {...} // fonction 1
template <typename T> void f(T, T) {...}           // fonction 2
template <typename T> void f(T, char) {...}         // fonction 3
template <typename T> void f(T, double) {...}       // fonction 4
void f(char, double) {...}                         // fonction 5
void f(int, double) {...}                          // fonction 6
```

et soient les déclarations de variables suivantes :

```
char c = 'A';
int i = 1;
float x = 2;
double y = 3;
```

Pour chacun des appels ci-dessous, indiquer soit quelle fonction est appelée, soit que l'appel est ambigu.

IMPORTANT

- Il n'est PAS demandé de justifier vos réponses ou de proposer un quelconque correctif
- Barème appliqué : +0.075 point pour une réponse correcte; 0 point en cas d'absence de réponse; -0.075 point pour une réponse incorrecte

1) f(i, c);

fonction 3 ✓

2) f(c, c);

ambigu ✓

3) f(i, x);

fonction 1 ✓

4) f<>(i, y);

fonction 4 ✓

5) f<double>(c, c);

fonction 3 ✓

6) f(i, (double)x);

fonction 6 ✓

6

0.5

b) (0.5 pts)

Soient les définitions de fonctions suivantes :

```
template <typename T> void f(T, T, char) {...} // fonction 1
template <typename T> void f(T, int, char) {...} // fonction 2
template <typename T> void f(T, int, int) {...} // fonction 3
template <typename T> void f(T, int, double) {...} // fonction 4
void f(char, int, short) {...} // fonction 5
void f(char, int, long) {...} // fonction 6
void f(char, int, double) {...} // fonction 7
void f(int, int, double) {...} // fonction 8
```

et soient les déclarations de variables suivantes :

```
char c = 'A';
short s = 1;
int i = 2;
float x = 3;
```

Pour chacun des appels ci-dessous, indiquer soit quelle fonction est appelée, soit que l'appel est ambigu.

IMPORTANT

- Il n'est PAS demandé de justifier vos réponses ou de proposer un quelconque correctif
- Barème appliqué : +0.125 point pour une réponse correcte; 0 point en cas d'absence de réponse; -0.125 point pour une réponse incorrecte

- | | |
|------------------|--------------|
| 1) f(c, i, x); | fonction 7 ✓ |
| 2) f(s, s, x); | ambigu ✓ |
| 3) f(i, s, x); | fonction 8 ✓ |
| 4) f<>(c, i, s); | fonction 3 ✓ |
- 4

0.3

c) (0.3 pts)

Soient les deux classes génériques suivantes :

```
template <typename T = int> class A {...};  
template <typename T, typename U, int n = 10> class B {...};
```

Pour chacune des instanciations ci-dessous, dire si celle-ci est correcte ou non.

IMPORTANT

- Il n'est PAS demandé de justifier vos réponses ou de proposer un quelconque correctif
- Barème appliqué : +0.1 point pour une réponse correcte; 0 point en cas d'absence de réponse; -0.1 point pour une réponse incorrecte

1) B<float*, float> b;

Correcte ✓

2) size_t size = 20;
B<int, array<float*, size>> b;

Incorrecte ✓

3) B<float, vector<A<>>, 2*10> b;

Correcte ✓

3

0.8

Question 4 (1 point)

Compléter les 2 parties notées *<à compléter>* du programme ci-après, de telle sorte que celui-ci affiche à l'exécution :

nombre d'occurrences = 4
nombre d'occurrences = 3

```
#include <algorithm>
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <vector>
using namespace std;

<à compléter 1>

template <typename T>
size_t nbOcc(const vector<T>& v, const Intervalle<T>& intervalle) {
    return (size_t) count_if(<à compléter 2>);
}

int main() {
    // Nombre de valeurs dans V1 comprises dans l'intervalle I1 semi-ouvert [1; 3[
    const V<int> V1{0, 1, 2, 3, 4, 3, 2, 1, 0};
    const I<int> I1{1, 3};
    cout << "nbre d'occurrences = " << nbOcc(V1, I1) << endl;

    // Nombre de valeurs dans V2 comprises dans l'intervalle I2 semi-ouvert ["A"; "D"]
    // ... ou dit autrement, nombre de prénoms commençant par A, B ou C
    const V<const char*> V2{"Albert", "Jack", "Bob", "Dylan", "Charles"};
    const I<const char*> I2{"A", "D"};
    cout << "nbre d'occurrences = " << nbOcc(V2, I2) << endl;

    return EXIT_SUCCESS;
}
```

IMPORTANT

- Hormis les 2 parties notées *<à compléter>*, aucune ligne de code ne doit être ajoutée, modifiée ou supprimée du code proposé ci-dessus.
- La classe *Intervalle* doit obligatoirement être implémentée sous la forme d'une **classe foncteur générique**.

Votre réponse : $\langle \text{à compléter} \rangle$

```
template <typename T>  
using V = vector<T>;
```

```
template <typename T>  
class Intervalle {
```

public:

```
    Intervalle(const T& min, const T& max);
```

```
    bool operator() (const T& element) const;
```

private:

```
    const T min;
```

```
    const T max;
```

```
};
```

```
template <typename T>
```

```
Intervalle<T>::Intervalle(const T& min, const T& max) : min(min), max(max) {
```

```
template <typename T>
```

```
bool Intervalle<T>::operator() (const T& element T ele) const {  
    return element >= min && element <= max;
```

```
}
```

Faux! la donnée
précisait intervalle semi-ouvert

```
template <>
```

```
bool Intervalle<const char*>::operator() (const char* element élément) const {  
    return *element >= *min && *element <= *max;
```

```
}
```

même erreur

```
template <typename T>  
using I = Intervalle<T>;
```

Votre réponse (suite) :

< A compléte 2 >

return (size_t) count_if (v.begin(), v.end(), interval); ✓

0.75

Question 5 (0.75 point)

Que va afficher le programme ci-dessous si l'utilisateur saisit :

1) la valeur 1 ?

2) la valeur 2 ?

...

5) la valeur 5 ?

```
#include <cstdlib>
#include <exception>
#include <iostream>
#include <stdexcept>
using namespace std;

void onExit() {cout << "onExit" << endl;}
void onTerminate() {cout << "onTerminate" << endl;
                    exit(EXIT_FAILURE);}
void onUnexpected() {cout << "onUnexpected" << endl;}

void f(int n) throw (int) {throw (short) n;}
void g(int n) noexcept {throw n;}

int main() {

    atexit(onExit);
    set_terminate(onTerminate);
    set_unexpected(onUnexpected);

    int n;
    cout << "Donnez un entier : ";
    cin >> n; // On suppose la saisie utilisateur OK

    try {
        switch (n) {
            case 1: throw 1;
            case 2: throw &n;
            case 3: try { f(n); }
                    catch (short) {cout << "catch1" << endl;}
                    catch (...) {cout << "catch2" << endl;}
                    break;
            case 4: try { g(n); }
                    catch (int) {cout << "catch3" << endl;}
                    catch (...) {cout << "catch4" << endl;}
                    break;
            case 5: try { throw runtime_error("Oups!"); }
                    catch (const overflow_error& e) {cout << e.what() << endl; throw 'A';}
                    catch (const exception& e) {cout << e.what() << endl; throw 'B';}
                    catch (...) {cout << "catch5" << endl; throw 'C';}
                    break;
        }
    }
    catch (int&) {cout << "catch6" << endl; throw 'D';}
    catch (int) {cout << "catch7" << endl; throw 'E';}
    catch (const int&) {cout << "catch8" << endl;}
    catch (const int) {cout << "catch9" << endl;}
    catch (char& e) {cout << e << endl;}

    cout << "Fin main" << endl;

    return EXIT_SUCCESS;
}
```

Vos réponses :

- 1) catch b
on Terminate
on Exit ✓
- 2) on Terminate
on Exit ✓
- 3) on Unexpected
on Terminate
on Exit ✓
- 4) on Terminate ✓
on Exit
- 5) Oups
B
Fin main ✓
on Exit