5

Nom et prénom : Spinelli Isaïa

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 !)**

0.75

Question 1 (1 point)

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

```
00.00
23:50 + 01:10 = 01:00
01.00
```

est-ce normal qu'il est que un point et parfois deux ?

?

je ne vois pas la logique.

```
<à compléter>

int main() {
   cout << Temps() << endl;
   // Temps t = 23; // Sans les commentaires, serait refusé à la compilation
   Temps t1(23, 50);
   Temps t2(1, 10);
   cout << t1 << " + " << t2 << " = " << (t1 + t2) << endl;
   cout << (t1 += t2) << endl;
   return EXIT_SUCCESS;
}
```

explicit

(peut être par référence ou pas)

(spécialisation sans S)

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
- La classe *Temps* est supposée ne comporter que 2 données-membres : *heure* (0-23) et *minute* (0-59)
- **La classe *Temps* ne doit déclarer que les fonctions strictement nécessaires à l'exécution du *main* proposé, et rien d'autre[1] !**

  [1] cela signifie, en particulier, qu'aucun accesseur (sélecteur, modificateur) de même qu'aucune fonction-membre privée ne doit être proposé.
- Aucune fonction de la classe *Temps* ne doit être codée en ligne (*inline*)
- Il n'est pas demandé de commenter les fonctions de la classe *Temps,* ni de vérifier la validité des paramètres de celles-ci
- Appliquer l'encapsulation et éviter au maximum d'écrire du code redondant
- L'usage **des formes canoniques** est requis

Votre réponse :

```
#include <cstdlib>
#include <iomanip>
#include <iostream>

using namespace std;

class Temps {
    friend ostream& operator << (ostream& os, const Temps& rhs);
public:
explicit Temps (unsigned heure=0, unsigned minute=0);
Temps& operator += (const Temps& temps);
Temps operator + (const Temps& temps) const;
private:
  unsigned heure;
  unsigned minute;

};
```

pas forme canonique

Votre réponse (suite) :

```
ostream& operator << (ostream& os, const Temps& rhs)
{
    return os << setfill('0') << setw(2) << rhs.heure
              << ':' << setw(2) << rhs.minute;
}

Temps::Temps (unsigned heure, unsigned minute)
  : heure(heure), minute(minute) {}

Temps& Temps::operator += (const Temps& temps)
{
    minute += temps.minute;
    if (minute > 59)
    {   ++heure;
        minute %= 60;
    }
    heure = (heure + temps.heure > 23)? heure % 24 : heure;
    return *this;
}

Temps Temps::operator + (const Temps& temps) const
{   Temps t = *this;
    return t += temps;
}
```

OK... mais peut s'écrire de manière plus compacte

1.25

Question 2 (1.25 point)

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

```
K = 10
c1 = (1, 0)
c2 = (2, 3)
c1 = (2, 3)
```

```
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, int j = 0);
9     C&operator=(const C& c);
10    int getI() const;
11    int getJ() const;
12    static int getK() const;
13 private:
14    int i;
15    const int j;
16    static const int K = 10;
17 };
18
19 friend ostream& operator<<(ostream& os, const C& c) {
20    return os << "(" << c.i << ", " << c.j << ")";
21 }
22
23 C(int i, int j = 0) {
24    this->i = i;
25    this->j = j;
26 }
27
28 C&operator=(const C& c) {
29    return *this;
30 }
31
32 int getI() const {return i;}
33 int getJ() const {return j;}
34 static int getK() const {return K;}
35
36 int main() {
37    cout << "K = " << C::getK() << endl;
38    C c1(1), c2(2, 3);
39    cout << "c1 = " << c1 << endl
40         << "c2 = " << c2 << endl;
41    cout << "c1 = " << (c1 = c2) << endl;
42    return EXIT_SUCCESS;
43 }
```

Ce code contient toutefois diverses erreurs ou maladresses.

RÉÉCRIRE INTÉGRALEMENT le code proposé (à l'exception des lignes 1 à 4 et 35 à 43 qui ne contiennent aucune erreur) de manière à corriger toutes les erreurs / maladresses qu'il contient.

IMPORTANT

- La nature des 3 champs ("usuel" pour *i*, "usuel constant" pour *j* et "static const" pour *K*) 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 supplémentaires sont introduites

Votre réponse :

Votre réponse (suite) :

```
class C {
  friend ostream& operator << (ostream& os, const C& c);
public:
    C(int i, int j=0);
  C& operator = (const C& c);
  int getI() const;
  int getJ() const;
  static int getK();
private:
    int i;
    const int j;
    static const int K=10;
};

ostream& operator << (ostream& os, const C& c)
{   return os << "(" << c.i << "," << c.j << ")";
}
C::C(int i, int j)
  : i(i), j(j) {}

C& C::operator = (const C& c)
{    i = c.i;
  (int&) j = c.j;
   return *this;
}
int C::getI() const
{ return i;}
int C::getJ() const { return j;}
int C::getK() { return K;}
```

1.05

Question 3 (1.25 point)

0.45

a) **(0.45 point)**
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, int) {…}           // fonction 3
template <typename T> void f(T, double) {…}        // fonction 4
void f(int, double) {…}                            // fonction 5
void f(double, double) {…}                         // fonction 6
```

et soient les déclarations de variables suivantes :

```
char c = 'A';
int i = 1;
long j = 2;
double x = 3.0;
```

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, i);` ambigu ✓
2) `f(c, i);` 3 ✓
3) `f(c, c);` 2 ✓
4) `f(c, j);` 1 ✓
5) `f<>(i, x);` 4 ✓
6) `f<double>(i, i);` 3 ✓

0.2 b) **(0.4 point)**
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
void f(char, int, double) {…}                       // fonction 4
void f(int, int, int) {…}                           // fonction 5
void f(int, int, long) {…}                          // fonction 6
void f(int, int, double) {…}                        // fonction 7
```

et soient les déclarations de variables suivantes :

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

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 : +0.1 point pour une réponse correcte; 0 point en cas d'absence de réponse; –0.1 point pour une réponse incorrecte

1) `f(i, i, c);` ambigu ✓
2) `f(i, c, s);` ~~3~~ 5 wtf!
3) `f(s, i, s);` 3 ✓
4) `f(c, s, x);` 4 ✓

1er param. 2eme param 3eme param

2) {1,2,3,5,6,7} ∩ {toutes} ∩ {3,5}
exacte promo. numérique promo. num.

5 car pas générique

Just for you

0.4

c) **(0.4 point)**
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<int, int*> b1;` correcte ✓

2) `B<double, A<>, false> b2;` correcte ✓

3) `B<int, vector<A>, 1> b3;` incorrecte ✓

4) non const
`size_t size = 100;`
`B<array<int, size>, array<int, size>> b4;` incorrecte ✓

0.1

Question 4 (0.75 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 = 6
```

```
#include <algorithm>
#include <cstdlib>
#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 count_if(<à compléter 2>);
}

int main() {
   // Nombre de valeurs de VECTEUR comprises dans l'intervalle [1; 3]
   const V<int> VECTEUR{1, 2, 3, 4, 3, 2, 1};
   const I<int> INTERVALLE{1, 3};
   cout << "nombre d'occurrences = " << nbOcc(VECTEUR, INTERVALLE) << 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.

Votre réponse :

```
template <typename T, size_t N=7>
class V {
   vector<T> vecteur;
public:
template <typename X> V::V (const X& val) { vecteur = X; }
};

template <typename U>
class I {
 Intervalle<U> intervalle;
public:
template <typename Y> I::I (const Y& val)
{ intervalle = Y; }
};
```

Votre réponse (suite) :

```
template <typename T>
bool Egal (const T& v , const Intervalle<T& intervalle)
{
    return (v == intervalle.min || v == intervalle.max);
}
```

approche incorrecte
cf corrigé en classe

```
template <typename T>
using V = vector<T>;
template <typename T> class Intervalle;
template <typename T>
using I = Intervalle<T>;
```

compléter 2%

```
count_if (v.begin(), v.end(), Egal);
```

Intervalle

```
template <typename T> class Intervalle {
    constructeur ...
    bool operator () (const T& elem) ....
}
```

(Foncteur)

0.3

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;
                       throw 1; }

void f(int n) noexcept {throw (double) 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 'A';
         case 2: throw (const int*) &n;
         case 3: try { throw overflow_error("Oups!"); }
                 catch (const logic_error& e) { cout << "Mince!" << endl;
                                                cout << e.what() << endl;
                                              }
                 catch (const exception& e)   { cout << "Bon sang!" << endl;
                                                cout << e.what() << endl;
                                              }
                 break;
         case 4: try { f(n); }
                 catch (float) { cout << "catch 1" << endl; }
                 catch (...)   { cout << "catch 2" << endl; }
                 break;
         case 5: try { throw range_error("Aïe!"); }
                 catch (const exception e) { cout << e.what() << endl;
                                             throw (const int) 1; }
                 break;
      }
   }
   catch (int)           { cout << "catch 3" << endl; }
   catch (int&)          { cout << "catch 4" << endl; }
   catch (int*)          { cout << "catch 5" << endl; throw 1.; }
   catch (const int)     { cout << "catch 6" << endl; }
   catch (const int&)    { cout << "catch 7" << endl; }
   catch (const int*)    { cout << "catch 8" << endl; throw 2.; }
   catch (const double)  { cout << "catch 9" << endl; }

   cout << "Fin main" << endl;

   return EXIT_SUCCESS;
}
```

Vos réponses :

1) onTerminate
onExit ✓

FDP

2) ~~catch 5~~
~~onTerminate~~
~~onExit~~

catch 8
onTerminate
onExit

(const sur pointer → non modifiable)

exception
↑
runtime_error
↑
overflow_error

3) ~~Mince!~~
~~Oups!~~
~~Fin main~~
~~onExit~~

Bon sang!
Oups!
Fin main
onExit

4) ~~onUnexpected~~
~~onTerminate~~
~~onExit~~

throw A,B

onTerminate
onExit

noexcept

5) std::exception
catch 3
Fin main
onExit ✓