```
<à compléter>

int main() {
   cout << Date() << endl;
//    Date d1 = 1; // Sans les commentaires, serait refusé à la compilation
   Date d2(11, 2016);
   cout << d2++ << endl;
   cout << d2 << endl;
   cout << ++d2 << 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
- La classe *Date* est supposée ne comporter que 2 données-membres : *mois* (1-12) et *annee*
- **La classe *Date* ne devra comporter qu'une seule fonction *non* membre au plus**
- Aucune fonction de la classe *Date* ne doit être codée en ligne (inline)
- Il n'est pas demandé de vérifier la validité des paramètres des diverses fonctions de la classe *Date*
- N'implémenter que ce qui est strictement nécessaire à la résolution du problème
- 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<cstdlib>
#include<iostream>
#include<iomanip>
```
Manque using...

Attention à passer les paramètres en const !

```
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;
};
```

Dans le setfill on met un caractère et non une chaîne.

```
ostream& operator << (ostream& lhs, const Date& rhs){
    << setfill('0') << setw(2) << ← plus propre
    lhs << (mois > 9) ? " " : "0" << rhs.mois << "." << rhs.annee;
    return lhs;
}
```

Formulation pas des plus "propre" et fausse sans les parenthèses !

```
Date::Date(unsigned mois, unsigned annee)
: mois(mois), annee(annee) {}

Date & Date::operator ++(){
    if(++mois > 12){
        mois = 1;
        annee++;
    }
```

```
Date Date::operator ++(){
    return *this = mois < 12 ? Date(mois+1, annee) : Date(1, annee+1)
}
```

on doit renvoyer l'objet courant (*this) ⚠

Manque une partie !

Votre réponse (suite) :

à noter : le paramètre n'est pas nommé, si on avait mis (int n), on aurait eu un warning à la compilation indiquant que n n'est pas utilisé.

```
Date Date::operator++(int){
    Date temp = *this;
    ++*this;
    return temp;
}
```

✓

20,75/26

1.15

Question 2 (1.25 point)

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

```
c = 5
obj1 = (1, 2)
obj2 = (3, 4)
obj1 = (3, 4)
```

```
1 #include <cstdlib>
2 #include <iostream>
3 using namespace std;
4
5 class MaClasse {
6    friend ostream operator>>(ostream os, const MaClasse& obj) const;
7 public:
8    void MaClasse(int a, int b);
9    MaClasse operator=(const MaClasse& obj) const;
10   int getA() const;
11   int getB() const;
12   static int getC() const;
13 private:
14   int a;
15   const int b = 0;
16   static int c = 5;
17 }
18
19 friend ostream operator>>(ostream os, const MaClasse& obj) const {
20   return cout << "(" << a << ", " <<  b << ")";
21 }
22
23 void MaClasse(int a, int b) {
24   this->a = a;
25   this->b = b;
26 }
27
28 MaClasse operator=(const MaClasse& obj) const {
29   if (*this != obj) {
30      a = obj.a;
31      b = obj.b;
32      c = obj.c;
33   }
34   return this;
35 }
36
37 int getA() const {return a;}
38 int getB() const {return b;}
39 static int getC() const {return c;}
40
41 int main() {
42   cout << "c = " << MaClasse::getC() << endl;
43   MaClasse obj1(1, 2), obj2(3, 4);
44   cout << "obj1 = " << obj1 << endl
45        << "obj2 = " << obj2 << endl;
46   obj1 = obj2;
47   cout << "obj1 = " << obj1 << endl;
48   return EXIT_SUCCESS;
49 }
```

← faux car b est const

ce n'est pas le contenu qui est important, c'est la référence ! il faut donc enlever le déréférencement de this et ajouter un référencement à l'objet obj...

Ce code contient toutefois diverses erreurs ou maladresses.

RÉÉCRIRE INTÉGRALEMENT le code proposé (à l'exception des lignes 40 à 49 qui ne contiennent aucune erreur) de manière à corriger toutes les erreurs / maladresses qu'il contient.

IMPORTANT

- La nature des 3 champs ("usuel" pour *a*, "usuel constant" pour *b* et static pour *c*) 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 :

```
#include <cstdlib>
#include <iostream>
using namespace std;
class MaClasse {
    friend ostream& operator << (ostream& os, const MaClasse& obj);
public:
    MaClasse(int a, int b = 0);   (on doit respecter la volonté du programmeur, qui
                                   était d'initialiser le champ b par défaut à 0)
    MaClasse& operator = (const MaClasse& obj);
    int getA() const;
    int getB() const;
    static int getC();
private:
    int a;
    const int b;
    static int c;
};

int MaClasse::c = 5;

ostream& operator << (ostream& os, const MaClasse& obj) {
    return os << "(" << obj.a << ", " << obj.b << ")";
}
```

Votre réponse (suite) :

```
MaClasse::MaClasse(int a, int b)
    :a(a), b(b) {}

MaClasse& MaClasse::operator=(const MaClasse& obj){
    a = obj.a;
    ~~const~~
    (int&)b = obj.b;
    return *this;
}

int MaClasse::getA() const { return a; }
int MaClasse::getB() const { return b; }
int MaClasse::getC() { return c; }
```

< Quid du test ?

```
if(*this != &obj){
    a = obj.a;
    (int&)b = obj.b;
```

l'idée du test est d'éliminer la self affectation

23/25

0.75

Question 3 (1 point)

a) 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.
(NB : Il n'est pas demandé de justifier vos réponses)

1) `f(i, c);` fonction 3 ✓ (3 unique et plus spécialisée que 1)

2) `f(c, c);` ambigu ✓ (car 2 et 3 sont plus spécialisées que la 1)

3) `f(i, x);` fonction 1 ✓ (car seule correspondance exacte)

4) `f<>(i, y);` fonction 4 ✓ (et non 6 à cause de <>)

5) `f<double>(c, c);` fonction 3 ✓ (car unique et plus spécialisée que 1)

6) `f(i, (double)x);` fonction 6 ✓

7) `f<double, double>(x, x);` ~~ambigu~~ 1 (car deux types forcés)

la seule fonction qui peut être forcée sur 2 types c'est la première. (la 2, par exemple, ne peut forcer qu'un type, le T)

b) Soient les définitions de fonctions suivantes : quand on peut faire des promotions numériques, on reste dans la recherche dans les fonctions génériques.

```
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(char, int, long) {...}                         // fonction 5
void f(int, int, double) {...}                        // fonction 6
```

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.
(NB : Il n'est pas demandé de justifier vos réponses)

1) `f(c, i, s);` ~~ambigu~~ 3 (promotion numérique short → int)

2) `f(c, i, x);` ~~ambigu~~ 4

3) `f<>(c, i, x);` ambigu ✓

4) `f(i, s, x);` fonction 6 ✓ $\{2,3,6\} \cap \{2,3,4,5,6\} \cap \{4,6\} = \{6\}$

on doit raisonner paramètre par paramètre

5) `f(s, s, x);` ambigu ✓ $\{1,2,3\} \cap \{1\} \cap \{4,6\} = \emptyset \Rightarrow$ ambigu

pas le 6, car on n'a pas le type exacte au départ et après il y a correspondance exacte dans les générique, du coup on ne revient pas aux fonctions usuels.

0.45

Question 4 (1 point)

En exploitant au maximum la généricité et en écrivant le moins de code possible, compléter la partie notée *<à compléter>* du programme ci-après, de telle sorte que celui-ci affiche à l'exécution :

```
v_int = [1,2,3]
l_int = [1,2,3]
d_int = [1,2,3]
v_boissons = [0,2]
l_boissons = [0,2]
d_boissons = [0,2]
```

```
#include <cstdlib>
#include <iostream>
#include <list>
#include <deque>
#include <vector>

using namespace std;

enum class Boisson {COCA, FANTA, SPRITE};

<à compléter>

int main() {

   vector<int> v_int{1, 2, 3};
   list<int> l_int{1, 2, 3};
   deque<int> d_int{1, 2, 3};
   cout << "v_int = " << v_int << endl;
   cout << "l_int = " << l_int << endl;
   cout << "d_int = " << d_int << endl;

   vector<Boisson> v_boissons{Boisson::COCA, Boisson::SPRITE};
   list<Boisson> l_boissons{Boisson::COCA, Boisson::SPRITE};
   deque<Boisson> d_boissons{Boisson::COCA, Boisson::SPRITE};
   cout << "v_boissons = " << v_boissons << endl;
   cout << "l_boissons = " << l_boissons << endl;
   cout << "d_boissons = " << d_boissons << endl;

   return EXIT_SUCCESS;
}
```

IMPORTANT

- Le code déjà fourni dans l'encadré ci-dessus ne doit en aucun cas être modifié
- En informatique, une *double-ended queue* (abrégée *deque*) est un type de données abstrait généralisant le concept de queue

Votre réponse :

~~string boisson[] = {COCA, FANTA, SPRITE};~~

```
template <typename T, template <typename> class CONTENEUR >
ostream& operator << (ostream& os ; const CONTENEUR <T>& o
```

l'idée y est mais pas la syntaxe

```
template <typename T, template <typename> class CONTENEUR >
ostream& operator << (ostream& os ; const CONTENEUR <T> & obj){
    os << "[";
    auto fin_conteneur = CONTENEUR.end(); auto debut_conteneur = CONTENEU
    for (auto i = CONTENEUR.begin(); i != fin_conteneur; ++
        if (i != debut_conteneur) { os << ", " ; }
        os << *i ;
    }
    os << "]";
    return os;
}
```

obj ! obj ! ???

⚠ Ne fonctionnera pas pour enum class Boisson

une enum class (contrairement à une enum simple) nécessite qu'on cast explic en int)

0.45

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 << "Bye-bye!" << endl;}
void onTerminate()   { cout << "Good luck!" << endl;
                       exit(EXIT_FAILURE); }
void onUnexpected()  { cout << "Last but not least!" << endl;
                       throw 1; }

void f(int n) throw (long) { throw n; }
void g(int n) throw (int)  { throw (short)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 (int) 1;
         case 2 : throw &n;
         case 3 : try { f(n); }
                  catch (short) { cout << "catch 1" << endl; }
                  catch (...)   { cout << "catch 2" << endl; }
                  break;
         case 4 : try { g(n); }
                  catch (short) { cout << "catch 3" << endl; }
                  catch (...)   { cout << "catch 4" << endl; }
                  break;
         case 5 : try { throw runtime_error("Gloups!"); }
                  catch (const range_error& e) { cout << e.what() << endl; }
                  catch (const exception& e)   { cout << e.what() << endl; }
                  catch (...)                  { cout << "catch 5" << endl; }
                  break;
      }
   }
   catch (int&)        { cout << "catch 6"  << endl;
                         throw 'A'; }
   catch (int)         { cout << "catch 7"  << endl;
                         throw 'A'; }
   catch (const int&)  { cout << "catch 8" << endl; }
   catch (const int)   { cout << "catch 9" << endl; }
   catch (const char)  { cout << "catch 10" << endl; }

   cout << "Fin main" << endl;

   return EXIT_SUCCESS;
}
```

Vos réponses :

1) catch 6
Good luck!
Bye-bye!

2) Good luck!
Bye-bye!

3) ~~Last but no least!~~
~~catch 6~~
~~Good bye! luck!~~
~~Bye-bye!~~

catch 2
Fin main
Bye-bye!

4) Last but not least!
catch 6
Good luck
Bye-Bye!

5) Gloups!
Fin main
Bye-bye!