Gérer des intervalles de valeurs

En mathématiques, un intervalle est l'ensemble des variables comprises entre deux valeurs. Par exemple, l'intervalle des entiers [1, 5[(fermé à gauche et ouvert à droite) correspond aux valeurs 1, 2, 3, 4.

Le but de ces exercices va être d'écrire une représentation d'ensemble d'intervalles. Par exemple l'ensemble [0, 5[U [10, 15[correspond aux valeurs 0, 1, 2, 3, 4, 10, 11, 12, 13, 14.

Note : par symétrie avec les conteneurs standards, nous allons travailler sur des intervalles fermés à gauche et ouverts à droite.

En particulier, il faudra pouvoir gérer les unions et intersections d'intervalles :

- [0, 5[U [2, 7[= [0, 7[
- [0, 5[I [2, 7[= [2, 5[

Intervalle simple

Première version

Écrire une classe `Interval_1` permettant de gérer un intervalle simple d'entiers (`int`).

Construction et affectation

- construction à partir de deux valeurs ;
- assertion que les deux valeurs sont ordonnées ;
- sémantique de valeur : copiable, déplaçable (move).

Opérations de base

- intersection d'intervalle ;

- tester si une valeur appartient à un intervalle.

```
class Interval_1 {
    ...
private:
    int low_value{};
    int hight_value{};
};
```

Seconde version

Idem, avec template au lieu de `int`.

```
template<class T>
class Interval_1_bis {
    ...
private:
    T low_value{};
    T hight_value{};
};
```

Intervalle composé

Première version

Idem, écrire une classe `Interval_2`, mais ajout de l'union d'intervalles. Il n'est plus possible de conserver uniquement deux valeurs correspondant aux bornes.

Utiliser un vector pouvant recevoir plusieurs `Interval_1` défini précédemment. Il faut en particulier s'assurer de la cohérence et la simplification des données (l'union de deux intervalles peut produire deux intervalles ou un seul).

```
class Interval_2 {
    ...
private:
    std::vector<Interval_1> m_intervals{};
};
```

Seconde version

Utiliser un conteneur associatif (std::set) pour trier les intervalles. Note : deux intervalles disjoints sont ordonnables, deux intervalles non disjoints doivent être fusionnés pour former un seul intervalle.

```
class Interval_2_bis {
    ...
private:
    std::set<Interval_1> m_intervals{};
};
```

Ensemble de valeurs

Première version

Écrire une classe `Interval_3`, en changeant de représentation interne. Au lieu de conserver des paires de valeurs pour représenter des intervalles, utiliser un tableau de valeurs avec l'état courant.

- l'intervalle [-inf, inf[sera représenté en interne par un tableau vide ;
- l'intervalle [0, inf[sera représenté en interne par la valeur {0, true} ;
- l'intervalle [0, 1[sera représenté en interne par les valeurs {{0, true}, {1, false}} ;
- l'ensemble [0, 1[U [2, 3[sera représenté en interne par les valeurs {{0, true}, {1, false}, {2, true}, {3, false}}.

```
class Interval_3 {
    ...
private:
    std::vector<pair<int, bool>> m_intervals{};
};
```

Deuxième version

Idem, mais utiliser d'autres types que `bool`. Par exemple avec `char` :

```
interval<char> i;           // [-inf, +inf[ = ''
i[0] = 'A';                 // [-inf, 0[ = '', [0, +inf[ =
'A'
i[10] = 'B';                // [-inf, 0[ = '', [0, 10[ = 'A',
[10, +inf[ = 'B'
cout << i[5] << std::endl;  // 'A'
```

Écrire la classe correspondante :

- sémantique de valeur ;
- ajout de valeurs ;
- union et intersection ;
- tester une valeur.

```
class Interval_3_bis {
    ...
private:
    std::vector<pair<int, char>> m_intervals{};
};
```

Deuxième version

Idem, mais avec templates

```
template<class T, class U>
class Interval_3_bis {
    ...
private:
    std::vector<pair<T, U>> m_intervals{};
};
```

Chapitre précédent	Sommaire principal	Chapitre suivant