Objets Simples & Opérateurs

Pour valider la description d'un circuit intégré, il est nécessaire de passer par une étape de simulation (cf. `asimut`). Une variable à l'intérieur du simulateur est un booléen, mais dans certains cas on peut ne pas connaître sa valeur. Le simulateur introduit donc un nouveau type de variable pouvant prendre trois valeurs: **0**, **1** ou **U** (`Undefined` ou encore `non-définie`).

Au cours de ce TME nous allons créer ce nouveau type (ou classe) que nous appellerons `BoolValue`.

Spécification de `BoolValue`

La classe `BoolValue` va contenir un seul attribut `_value` qui sera de type entier et ne pourra prendre que trois valeurs: 0, 1 ou 2 ayant respectivement la signification `Zero`, `One` et `Undefined`.

Plutôt que travailler avec des constantes numériques, nous allons utiliser des constantes nommées. On réalise cela avec un `enum`, comme ci-après:

```
class BoolValue {
  public:
    enum Value { Zero=0, One=1, Undefined=2 };
  public:
    // ...
};
```

A l'extérieur de la classe on les référencera avec la syntaxe `BoolValue::Undefined` et à l'intérieur, simplement comme `Undefined`.

La classe `BoolValue` possèdera les fonctions membres suivantes:

Constructeurs: (*CTOR*)

- `BoolValue()`, le constructeur par défaut, devra affecter la valeur `Zero`.
- `BoolValue{{{int`, un contructeur à partir d'un entier ordinaire. Tout entier de valeur supérieure ou égale à 2 sera considéré comme `Undefined`.
- `BoolValue(const BoolValue&)`, un constructeur par copie.

Destructeur: (*DTOR*)

- `~BoolValue()`, le destructeur (unique).

Accesseurs: (*Accessors*)

- `print(std::ostream&)`, une fonction d'affichage.
- `toInt()`, une fonction de conversion vers un entier.

Modifieur: (*Mutators*)

- `fromInt(int)`, affectation depuis un entier.

En plus des fonctions membres listées çi dessus, on ajoute les fonctions non-membres suivantes permettant de réaliser des opérations logiques:

- `BoolValue non(const BoolValue&)`, négation logique.
- {{{BoolValue? ou (const BoolValue?&, const BoolValue?&), *ou* logique.
- {{{BoolValue? et (const BoolValue?&, const BoolValue?&), *et* logique.

La table de vérité des fonctions négation et *ou* vous est fournie çi-après.

Question 1

Implanter la class `BoolValue` telle que décrite précédemment. La tester à l'aide du petit programme de test suivant:

[[Image(CheckOuCode?-1.png,70%,align=center)]