

## Langage C++ - TP 1

Le but du TP est de modéliser de façon très simple des portes logiques puis de les connecter entre elles dans un réseau qui sera exécuté et permettra de simuler de façon très simple des petits circuits.

Il convient d'adopter dès à présent des conventions de programmation *propre* :

- Découpez votre code en déclaration dans un fichier `.h` et implémentation dans un fichier `.cc` ou `.cpp`. Par exemple `circuit.h` et `circuit.cc`. Mettez le tout dans un répertoire créé pour l'usage.

- Utilisez une convention de nommage, les noms de classes seront par exemple tous en `ClassName` (aussi appelé CamelCase majuscule), les noms de fonctions du type `function_name()`, et les champs de classe en `m_class_field`.

- Utilisez une indentation unique dans tous les fichiers.

- Commentez votre code en anglais. Commentez les fonctions dans les déclarations dans le `.h`. Commentez les gros blocs de code d'implémentation si nécessaire. Utilisez des noms de classes et de fonctions en anglais.

Passons à présent à ce premier TP.

### Exercice 1

Commencez par proposer une modélisation orientée objet des portes logiques. En particulier chaque type de porte logique devra être représenté sous forme d'une classe et une méthode commune à tous les types de porte devra permettre d'y passer d'y envoyer un signal et d'en récupérer la sortie. Cette méthode devra être déclaré comme `bool process(bool, bool)`. En effet ne sont considéré que les portes logiques à deux entrées et une sortie.

Pour rappel, le C et le C++ proposent des opérateurs booléens, que l'on peut directement utiliser avec des booléens de type `bool` en C++:

```
a & b    // ET
e ^ f    // XOR
~ g      // NOT
```

Vous implémenterez pour l'instant au minimum les composants `GateAND` et `GateXOR`. Écrivez un test, `test-1.cc` (ou `.cpp`) pour vérifier les fonctions `process()` des portes :

```
GateAnd a;
cout << "0 AND 1: " << a.process(false, true) << endl;
```

### Exercice 2

Tentez de connecter dans un fichiers `test-2.cc` (ou `.cpp`) deux ou trois portes logiques et vérifiez la sortie :

```
GateAnd a;
GateXor b;
cout << "1 AND (0 XOR 1): "
    << a.process(1, b.process(0, 1)) << endl;
```

### *Exercice 3*

En utilisant les variables statiques, affichez à chaque création de porte logique, le nombre de portes actuellement en service.

### *Exercice 4*

Modifiez vos classes pour que les méthodes `process()` prennent des tableaux de booléens en entrée plutôt qu'une liste d'arguments. De même, faites en sorte que ces méthodes retournent un tableau de booléens en sortie.

### *Exercice 5*

En utilisant les portes ET et OU déjà définies, créez une classe `GateHADD` qui sera un demi-additionneur logique à deux entrées et deux sorties :

```
bool* GateHADD::process(bool* input) {
    // ...
    m_output[0] = xor.process(input[0], input[1]);
    // ...
    return m_output;
}
```

*Mercredi 15 septembre 2010*