

Langage C++ - TP 2

Dans le TP précédent nous avons créé quelques portes logiques et les avons manuellement testé. Nous allons à présent créer des objets plus générique et profiter des aspects orienté objet du C++. Dans ce TP, les éléments de base considérés seront des composants d'un circuit électronique.

Créez un répertoire `tp-2` dans lequel vous éditez les fichiers `circuit.h`, `circuit.cc` (ou `.cpp`) et `test.cc`. Respectez toujours les consignes de propreté du code données dans le TP précédent.

Exercice 1

Nous travaillons à présent avec des composants électroniques pouvant gérer plusieurs sorties. La méthode `process()` devient alors `bool* process(bool*)` qui prend un tableau de booléens en entrée et retourne un tableau de booléen en sortie. Pour ne pas avoir à créer le tableau de booléens pour chaque composant et à chaque appel de `process()` je vous conseille d'utiliser les bouts de code suivants.

```
class Gate
{
public:
    Gate(int output_size);
    virtual ~Gate();
    // ...
protected:
    bool* m_output;
};
// ...
Gate::Gate(int output_size)
{
    m_output = new bool[output_size];
}
Gate::~~Gate()
{
    if (m_output != NULL)
        delete [] m_output;
}
// ...
class GateAND : public Gate
{
public:
    GateAND();
}
// ...
GateAND::GateAND()
    : Gate(1)
{
}
```

Exercice 2

Créez à présent une classe `Connection` qui va établir des liens entre deux composants. Une connexion ne s'effectue que depuis la sortie

d'un composant vers l'entrée d'un autre composant. Elle doit être définie comme deux paires de combinaison d'un composant et d'un index d'entrée (ou de sortie).

Exercice 3

Le type de composant `Circuit` est lui-même composé d'autres composants. Commencez par le déclarer en faisant en sorte qu'il soit un conteneur de `Connection`. Utilisez la classe patron `std::vector<T>` dont l'usage est illustré ci-dessous pour pouvoir facilement créer des méthodes permettant d'ajouter des connexions.

```
#include <iostream>
#include <vector>
using namespace std;
int
main()
{
    vector<int> v;
    v.push_back(123);
    v.push_back(38);
    int sum = v[0] + v[1];
    cout << v[0] << " + " << v[1] << " = "
         << sum << endl;
    v.erase(); // Efface tout le tableau
    cout << "v contient " << v.size()
         << " éléments" << endl; // 0
    return 0;
}
```

Exercice 4

Il manque encore les entrées et les sorties dans notre agrégat de composants. Définissez-les comme des `vector` de `Slot`.

Exercice 5

Vous pouvez à présent définir `Circuit::process()`. Pour cela vous devez calculer la sortie à partir des entrées en prenant en compte toutes les connexions.

Exercice 6

Testez votre classe en créant un demi-additionneur dans un circuit.

Exercice 7

Créez un fonction pour construire des additionneur complets.

Exercice 8

Utilisez cette fonction pour construire des additionneurs à nombre de bits variable.

Mercredi 22 septembre 2010