CR TME 6

Objectif du TME : Réaliser un parseur XML d'une Netlist.

Fonctions à réaliser :

- Term* Term::fromXml(Cell*, xmlTextReaderPtr)
- Instance* Instance::fromXml(Cell*, xmlTextReaderPtr)
- Net* Net::fromXml(Cell*, xmlTextReaderPtr)
- bool Node::fromXml(Net*, xmlTextReaderPtr)

Quelques modifications ont étés faites par rapport au code du TME 4-5 pour plus de lisibilité et ajouts d'une ou deux méthodes précédemment oubliées.

Term* Term::fromXml(Cell*, xmlTextReaderPtr) :

On vérifie que le nom du noeud XML actuel correspond au tag term voulu. Si c'est le cas, on récupère les attributs du Term.
On vérifie que l'attribut name n'est pas vide, puis on crée le nouveau Term. On récupère puis set la position et on retourne le Term précédemment créé. La boucle du Cell:fromXml() se chargera de rappeler la fonction pour les term suivants.

```
90      Term* Term::fromXml(Cell* c, xmlTextReaderPtr reader){
91          const xmlChar* nodeTag  = xmlTextReaderConstString        ( reader, (const xmlChar*)"term" );
92          const xmlChar* nodeName = xmlTextReaderConstLocalName     ( reader );
93          Term* term = NULL;
94
95          //vérification du nom du node actuel par rapport au Tag demandé
96          if(nodeTag == nodeName){
```

```
97
98              //récupération des attributs
99              std::string termName = xmlCharToString( xmlTextReaderGetAttribute( reader, (const xmlChar*)"name" ) );
100             std::string termDirection = xmlCharToString( xmlTextReaderGetAttribute( reader, (const xmlChar*)"direction" ) );
101             Direction d = term->toDirection(termDirection);
102
103             //test si nom du term est vide
104             if(termName.empty()) return NULL;
105
106             //création du term
107             term = new Term(c, termName, d);
```

```
109             //ajout au term de la position
110             int termX = 0;
111             int termY = 0;
112             xmlGetIntAttribute( reader, "x", termX );
113             xmlGetIntAttribute( reader, "y", termY );
114             term->setPosition(termX, termY);
115         }
116         return term;
117     }
```

Instance* Instance::fromXml(Cell*, xmlTextReaderPtr) :

Exactement le même principe que pour Term::fromXml(). L'unique différence est que l'on récupère la mastercell de l'instance dans le document xml au lieu de la direction du Term.

```
Instance* Instance::fromXml(Cell* c, xmlTextReaderPtr reader){
    const xmlChar* nodeTag  = xmlTextReaderConstString        ( reader, (const xmlChar*)"instance" );
    const xmlChar* nodeName = xmlTextReaderConstLocalName     ( reader );

    Instance* instance = NULL;

    if(nodeTag == nodeName){

        std::string instanceName = xmlCharToString( xmlTextReaderGetAttribute( reader, (const xmlChar*)"name" ) );
        std::string mastercellName = xmlCharToString( xmlTextReaderGetAttribute( reader, (const xmlChar*)"mastercell" ) );

        if( instanceName.empty() or mastercellName.empty() ) return NULL;

        Cell* mastercell = mastercell->find(mastercellName);
        instance = new Instance(c, mastercell, instanceName);

        int instanceX = 0;
        int instanceY = 0;
        xmlGetIntAttribute( reader, "x", instanceX );
        xmlGetIntAttribute( reader, "y", instanceY );
        instance->setPosition(instanceX, instanceY);
    }
    return instance;
}
```

Net* Net::fromXml(Cell*, xmlTextReaderPtr) :

Fonctionnement légèrement différent car l'élément Net possède un ou plusieurs éléments fils de type Node.

```
<net name="a" type="External">
    <node term="a"                    id="0"/>
    <node term="i0" instance="xor2_1" id="1"/>
    <node term="i0" instance="and2_1" id="2"/>
</net>
```

En plus de vérifier que le nom du noeud correspond au tag net, on vérifie que le noeud actuel n'est pas la balise de fermeture du net en cours. On récupère les attributs et on crée le Net de la même façon que les deux fonctions précédentes.
On parcourt ensuite les éléments fils de type Node jusqu'à la balise de fermeture du Net actuel et on fait appel à Node::fromXml() pour les traiter un à un.

```
Net* Net::fromXml(Cell* c, xmlTextReaderPtr reader){
    Net* net = NULL;

    const xmlChar* nodeTag  = xmlTextReaderConstString        ( reader, (const xmlChar*)"net" );
    const xmlChar* nodeName = xmlTextReaderConstLocalName     ( reader );

    //vérification du nom du node actuel par rapport au Tag demandé + le node ne doit pas être une balise de fermeture
    if((nodeTag == nodeName) and !(xmlTextReaderNodeType(reader) == XML_READER_TYPE_END_ELEMENT))
    {
```

```
            //récupération des attributs du Net
            std::string netName = xmlCharToString( xmlTextReaderGetAttribute( reader, (const xmlChar*)"name" ) );
            std::string netType = xmlCharToString( xmlTextReaderGetAttribute( reader, (const xmlChar*)"type" ) );

            //création du Net
            Term::Type t = Term::toType(netType);
            net = new Net(c, netName, t);
```

```
            //Parcours des éléments fils du net jusqu'à la balise de fermeture
            while(!((nodeTag == nodeName) and (xmlTextReaderNodeType(reader) == XML_READER_TYPE_END_ELEMENT))){
                //on fait avancer le pointeur
                xmlTextReaderRead(reader);

                //Appelle de Node::fromXml() pour connecter le net au term.
                if( !(Node::fromXml(net, reader)) ) return NULL;
            }
        }
        return net;
    }
```

bool Node::fromXml(Net*, xmlTextReaderPtr) :

La fonction Node::fromXml() réalise la connexion entre un Net et un Term.
Deux cas possibles :

- Le node possède un attribut instance, alors il s'agit d'un terminal de cette instance.
- Le node ne possède pas d'attribut instance, alors c'est une connexion à un terminal de la cellule.

Récupération classique des attributs. On test si le nom de l'attribut instance est vide (et donc il n'existe pas) ou pas.
S'il n'existe pas, on récupère la Cell du Net actuel et on connecte le Net au Term de la cellule.
S'il existe, on récupère l'instance correspondante et connecte le Net au Term de celle ci.

```
    bool Node::fromXml(Net* net, xmlTextReaderPtr reader){
        const xmlChar* nodeTag  = xmlTextReaderConstString        ( reader, (const xmlChar*)"node" );
        const xmlChar* nodeName = xmlTextReaderConstLocalName     ( reader );
        int nodeId = -1;

        //vérification du nom du node actuel par rapport au Tag demandé
        if(nodeTag == nodeName){
```

```
            //récupération des attributs
            std::string nodeTerm = xmlCharToString( xmlTextReaderGetAttribute( reader, (const xmlChar*)"term" ) );
            std::string nodeInstance = xmlCharToString( xmlTextReaderGetAttribute( reader, (const xmlChar*)"instance" ) );
            xmlGetIntAttribute( reader, "id", nodeId );

            //on vérifie que le term n'est pas vide et que l'id du node est positif
            if(nodeTerm.empty() or nodeId<0) return false;
```

```
58            Term* term = NULL;
59
60            //on test l'existence de l'attribut instance
61            if(nodeInstance.empty()){
62                //on récupère la cell et on fait la connection
63                if( !(net->getCell()->connect(nodeTerm, net)) ) return false;
64                term = net->getCell()->getTerm(nodeTerm);
65                term->getNode()->setId(nodeId);
66            }else {
67                //on récuprère l'instance et on fait la connection
68                Instance* instance = net->getCell()->getInstance(nodeInstance);
69                if( !(instance->connect(nodeTerm, net)) ) return false;
70                term = instance->getTerm(nodeTerm);
71                term->getNode()->setId(nodeId);
72            }
73        }
74        return true;
75    }
```