

# Langage à Objets Avancé

Projet N° 02

January 19, 2022

## 1 Introduction

Un circuit combinatoire est un ensemble de portes logiques reliées entre elles pour répondre à une expression algébrique. Dans le projet , nous allons implémenter un ensemble de types et de fonctions qui nous permettront de créer un circuit combinatoire et de l'afficher. En plus , on doit pouvoir voir la progression de l'information pas à pas à chaque fois qu'elle franchit une porte logique et d'afficher sous forme textuelle les fonctions de sortie à l'aide des variables d'entrée. En outre , le circuit peut etre généré à partir de son expression textuelle, sauvé dans un fichier pour le lire ensuite.

## 2 Les classes

### 2.1 La classe "Gate"

Nous avons défini une nouvelle entité Gate , qui est la classe mère de toutes les autres classes qui définissent les portes logiques. Elle est caractérisé par:

- string name : un nom( qui est sur trois caractères , par exemple pour une porte "And" ça sera "AND" , et pour une porte "Nand" , ça sera "NAN" etc..)
- vector <Gate \* >\* : En plus du nom, une Gate possède un vecteur d'entrées (On peut avoir plus de deux entrées ) , ce vecteur (qui est un pointeur vers) un tableau qui contient des pointeurs vers des objets de type Gate
- bool valeurBooleenne : Une Gate possède une valeur de sortie qui sera calculée en fonction de la porte logique.

De la classe Gate héritent plusieurs classe qui sont :

1. AndGat : réalise AND( a1, a2, .. an) et retourne : a1 && a2 && .. && an
2. NandGate : réalise NAND( a1, a2, .. an) et retourne : ! ( a1&& a2 && ..&& an )
3. OrGat : réalise OR( a1, a2, .. an) et retourne : a1 — a2 OR .. OR an
4. NorGate : réalise AND( a1, a2, .. an) et retourne : !( a1 OR a2 OR ..OR an )
5. NegateGat : réalise NEGATE(a) et retourne !a
6. XorGate : réalise Xor(a,b) et retourne : !ab OU !ba
7. XnorGat : réalise Xor(a,b) et retourne : ! (!ab OR !ba)\*
8. InputGate : Une classe particulière des sous classes de Gate, elle possède un constructeur qui initialise le nom de l'input et sa valeur

9. OutputGate : Une classe particulière des sous classes de Gate, elle possède un constructeur qui initialise le nom de l'input et sa valeur

Les classes citées en dessus , redéfinissent la méthode `getValeurBooleenne()` de la classe mère, et calculent la sortie en fonction du type. La construction des objets , se passe en appelant le constructeur de la classe mère pour instancier une Gate avec le bon nom , puis l'initialisation du vecteur des entrées.

## 2.2 La classe "Circuit"

Cette classe représente le coeur du projet, et qui possèdent les méthodes permettant de répondre aux spécifications demandées . La création du circuit initialise le vecteur des entrées , le vecteur des portes logiques ainsi le vecteur de sortie. La classe "Circuit " possède comme données :

1. `vector <InputGate * >* inputs` : qui est le tableau contenant les entrées du circuit
2. `vector<OutputGate*>* outputs` : qui est le tableau de sorties du circuit ( dans l'exemple il y'avais qu'une sortie mais il peut y'avoir plusieurs)
3. `vector <Gate*>* gates` : contient toutes les portes logiques qui constituent le circuit
4. `vector <vector <char>* >* affichageCircuit` : le tableau qui représente le circuit logique, qui est un tableau bidimensionnel de caractères : qui peuvent être des "-" ou des "\*" ou des "+" ou bien des caractères alphabétique pour représenter les portes logiques"
5. `vector <vector <Gate*>* >* simulationPasParPas` : nous permet de faire la simulation en mode pas à pas : A chaque étape ,on garde le vecteur des gates qu'on a pour pouvoir voir la progression de l'information

La classe "Circuit" offre une interface aux autres classes en mettant les méthodes essentielles en publiques et en cachant les autres (nécessaires pour le fonctionnement des autres méthodes mais qui ne doivent pas nécessairement être vues par l'utilisateur) et qui sont :

1. `void afficheCircuit() const` : cette méthode permet d'afficher le circuit (représentation textuelle) que nous avons représenté sous forme d'un tableau bidimensionnel de caractères.
2. `void simulation()` Permet la simulation en mode pas à pas
3. `void changerValeursDesPortesEntree()` ça nous permet de Changer les valeurs des portes d'entrée
4. `Circuit* expressionTextuelleToCircuit(const string expressionTextuelle)` Construire un circuit à partir d'une représentation textuelle et le renvoyer
5. `void sauvegarderCircuitDansFichier(const string nomFichier) const` : Permet de sauvegarder un circuit dans un fichier ayant comme nom `nomFichier`
6. `string afficherSousFormeTextuelle() const` : permet d'afficher le circuit sous la représentation textuelle

Les autres méthodes sont mises en privé , et sont appelées lors de la création du circuit, faisant des traitements necessaire à l’affichage du circuit.

1. `vector <Gate*>* ajoutInputs()` :Une methode qui recherche les positions des caractères '+' et '\*' dans la dernière ligne du vecteur qui stoke l’affichage du circuit et ajoute aux memes positions dans la ligne en dessous le caractère '—'
2. `void ajoutCheminsApresInputs()` : Créer le chemin près avoir rajouté un input ie: (mettre les '—', '\*', '+' dans le bon endroit
3. `void ajoutNomsOperationsLogiques(const vector <Gate*>* portesLogiquesAjouter-Affichage)` : mettre les bonnes portes logique au bon endroit : en mettant le nom représenté sur trois caractères
4. `void ajoutCheminsApresOperationsLogiques()`
5. `void ajoutAsterisquesApresChemins(unsigned int level)`
6. `vector <Gate*>* trouverLesPortesLogiquesSuivantes(const vector<Gate*>* portesLogiquesPrecedentes);`

### 2.3 La classe Main

C’est la classe principale dans laquelle on appelle les méthodes (ci dessus ) correspondantes aux traitements qu’on souhaite effectuer, on a un menu de la forme :

- Afficher le circuit : appelle la méthode : `afficheCircuit()`
- Simulation en mode pas a pas, appelle la méthode : `simulation()`
- Changer les valeurs des portes d entree : `void changerValeursDesPortesEntree()`
- Afficher sous forme textuelle les fonctions de sortie a l’aide des variables d’entrée
- Synthétiser un circuit à partir d’une expression textuelle : `Circuit* expression-TextuelleToCircuit(const string expressionTextuelle)`
- Sauver un circuit dans un fichier
- Relire un circuit qui est dans un fichier

### 2.4 Code source!

Lien vers le code source du projet : [Projet CPP - Sujet 02 - Ait benali Faycal / Leonard Namolaru](#)

### 2.5 Exécution :

La compilation et l’exécution du programme se passent avec les commandes ”make all” et make test” comme suit :



```

8- Quitter
Votre choix : 2

*****
Bonjour et bienvenue au simulateur de circuit combinatoire !
La simulation sera effectuée en mode pas à pas, c'est-à-dire qu'à chaque pas l'information franchit au plus une porte,
et durant toutes les étapes vous pourrez voir dans l'affichage la progression de l'information
*****

a:0 --*--*-----*--*-----
b:0 --+--+*-----+--+*-----
c:0 --+--+*--+*-----+--+*-----
d:0 --+--+*--+*--+*-----+--+*-----
e:0 --+--+*--+*--+*--+*-----+--+*-----
f:0 --+--+*--+*--+*--+*--+*-----

*****
a : 0   b : 0   c : 0   d : 0   e : 0   f : 0
*****
      | | | | | | | | | |
      OR AND OR AND AND AND OR OR
*****
OR (a,b) : 0
*****
OR (a,b) : 0   AND(a,b) : 0
*****
OR (a,b) : 0   AND(a,b) : 0   OR (c,d) : 0
*****
OR (a,b) : 0   AND(a,b) : 0   OR (c,d) : 0   AND(c,d) : 0
*****
OR (a,b) : 0   AND(a,b) : 0   OR (c,d) : 0   AND(c,d) : 0   AND(e,f) : 0
*****
OR (a,b) : 0   AND(a,b) : 0   OR (c,d) : 0   AND(c,d) : 0   AND(e,f) : 0   AND(a,b) : 0
*****
OR (a,b) : 0   AND(a,b) : 0   OR (c,d) : 0   AND(c,d) : 0   AND(e,f) : 0   AND(a,b) : 0   OR (a,f) : 0
*****
*****

```

Pour changer les valeurs des entrées (les modifier) tapez 3 .On voit bien après l'affichage du circuit que les entrées ont été bien changées:

```

**** MENU : SIMULATEUR DE CIRCUIT COMBINATOIRE ****
1- Afficher le circuit
2- Simulation en mode pas à pas
3- Changer les valeurs des portes d'entrée
4- Afficher sous forme textuelle
5- Synthétiser un circuit à partir d'une expression textuelle
6- Sauvegarder un circuit dans un fichier
7- Relire un circuit qui est dans un fichier
8- Quitter
Votre choix : 3
Nom de l'entrée : a ; Valeur : 0
Nouvelle valeur [1 / 0] : 1

Nom de l'entrée : b ; Valeur : 0
Nouvelle valeur [1 / 0] : 0

Nom de l'entrée : c ; Valeur : 0
Nouvelle valeur [1 / 0] : 1

Nom de l'entrée : d ; Valeur : 0
Nouvelle valeur [1 / 0] : 0

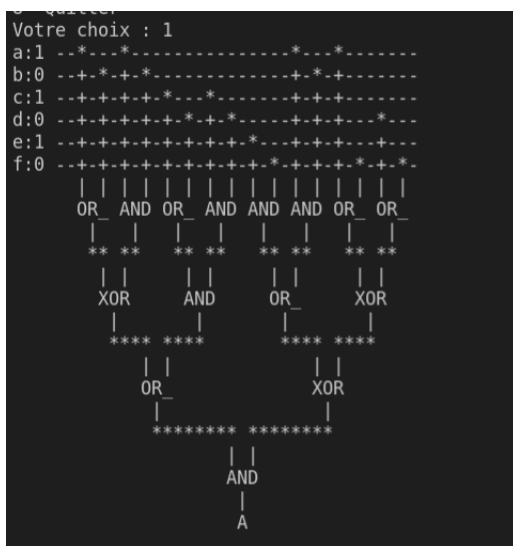
Nom de l'entrée : e ; Valeur : 0
Nouvelle valeur [1 / 0] : 1

Nom de l'entrée : f ; Valeur : 0
Nouvelle valeur [1 / 0] : 0

**** MENU : SIMULATEUR DE CIRCUIT COMBINATOIRE ****

```

Après exécution :



Pour avoir l'expression textuelle du circuit , tapez 4 , le résultat de la première exécution est :

```

*** MENU : SIMULATEUR DE CIRCUIT COMBINATOIRE ***
1- Afficher le circuit
2- Simulation en mode pas 0 pas
3- Changer les valeurs des portes d'entr e
4- Afficher sous forme textuelle
5- Synth tiser un circuit   partir d'une expression textuelle
6- Sauvegarder un circuit dans un fichier
7- Relire un circuit qui est dans un fichier
8- Quitter
Votre choix : 4

A=AND(OR_(XOR(OR_(a,b),AND(a,b)),AND(OR_(c,d),AND(c,d))),XOR(OR_(AND(e,f),AND(a,b)),XOR(OR_(a,f),OR_(d,f))))

```

Pour cr  er un circuit , avec une expression textuelle , on clique sur 5, voici les r  sultat d'ex  cution sur un exemple d'un circuit simple.

```

8- Quitter
Votre choix : 5
Synth tiser un circuit a partir d'une expression textuelle

Veuillez respecter les regles suivantes :
1 - Veuillez ne pas mettre des espaces dans l'expression textuelle.
2 - Une porte logique peut avoir comme entr  es deux entr  es du circuit ou deux entr  es qui sont des portes logiques. Mais il n'est pas possible de transf  rer une entree du premier type et une seconde entree du second type..
3 - Noms des portes logiques utilisables : xor (ou XOR), or (ou OR,OR_,or_), and (ou AND), nor (ou NOR),
4 - xnor (ou XNOR,xno, XNO), nand (ou NAND,nan et NAN)
5 - Bien qu'un circuit puisse avoir plusieurs entr  e , il n'a qu'une sortie.
Expression textuelle : A=AND(OR(a,b),XOR(a,b))
*** MENU : SIMULATEUR DE CIRCUIT COMBINATOIRE ***

```

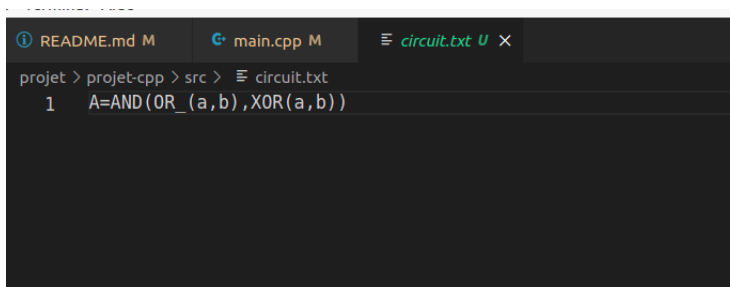
APr  s ex  cution , on affiche le circuit pour s'assurer que les changements on   t   pris en compte :

```

**** MENU : SIMULATEUR DE CIRCUIT COMBINATOIRE ***
1- Afficher le circuit
2- Simulation en mode pas à pas
3- Changer les valeurs des portes d'entrée
4- Afficher sous forme textuelle
5- Synthétiser un circuit à partir d'une expression textuelle
6- Sauvegarder un circuit dans un fichier
7- Relire un circuit qui est dans un fichier
8- Quitter
Votre choix : 1
b:0 ----*---*--
a:0 --*--*--+--
      | | | |
      | | | |
      XOR OR_
      | |
      ** **
      | |
      AND
      |
      A

```

L'exécution de la sauvegarde du circuit dans un fichier génère le fichier circuit.txt , qui a le contenu suivant :



```

1  A=AND(OR_(a,b),XOR(a,b))

```

### 3 Limites du programme :

Bien que le programme réalise la majorité des fonctionnalités demandées , il contient des limites et quelques améliorations à faire , on cite les plus importantes : - La porte NEGATE , n'ayant qu'une seule entrée ne peut être utilisée. - Plusieurs entrées sont permises , mais le circuit ne peut avoir qu'une sortie finale.