IN505 - Partie 1 : Cryptage et décryptage

Binôme :

- Samy Boumali - TD4
- Amine Atek - TD2

Notre programme s'organise en deux classes principales :

- **ArbreB -** Représentant un arbre binaire étiqueté par sa racine, sa taille et sa hauteur.
- **Sommet -** Représentant un sommet d'arbre binaire étiqueté par une Lettre, une Fréquence ainsi qu'un pointeur vers chacun de ses fils.

L'interface graphique s'organise quant à elle en quatre classes :

- **MainWindow -** Représentant la fenêtre principale.
- **MainMenu -** Représentant le layout de boutons de l'interface.
- **DessineArbre -** Zone de dessin contenant l'algorithme de représentation d'arbre binaire.
- **Context -** Classe Singleton permettant de coordonner la fenêtre, l'arbre et les boutons.

Voici ci-après un aperçu des attributs et méthodes de des deux classes principales susmentionnées (cf. fichiers headers) :

```
/**
 * Représente un sommet d'arbre binaire étiqueté par une Lettre, une Fréquence ainsi qu'un pointeur vers chacun de ses fils.
 *
 * Attributs privés:
 * @lettre      char        Lettre caractérisant le sommet
 * @freq        float       Fréquence d'occurence de cette lettre
 * @taille      int         1 <= nombre de sommets <= 3
 * @filsg       Sommet*     Pointeur vers le fils gauche du sommet
 * @filsd       Sommet*     Pointeur vers le fils gauche du sommet
 *
 */
class Sommet
{
    private :
        char lettre;
        float freq;
        int taille;
        Sommet * filsg;
        Sommet * filsd;

    public :
        /**
        * Constructeurs:
        * - Paramètres par défaut
        * - Constructeur vide déclaré implicitement
        * - Constructeur par copie
        */
        Sommet(const char =' ',const float = 0, const int = 0);
        Sommet(Sommet &);
```

```
/**
 * Représente un arbre binaire étiqueté par sa racine, sa taille et sa hauteur.
 *
 * Attributs privés:
 * @racine      Sommet*     Racine de l'arbre binaire, permettant l'accès à tous les autres nœuds de l'arbre
 * @taille      int         Nombre de sommets que composent l'arbre binaire
 * @hauteur     int         Hauteur de l'arbre binaire
 */
class ArbreB
{
    private :
        Sommet * racine;
        int taille;
        int hauteur;
    public :
        /**
        * Constructeurs:
        * - Paramètres par défaut
        * - Constructeur vide déclaré implicitement
        * - Constructeur par copie
        */
        ArbreB(Sommet * = NULL , int = 0);
        ArbreB(ArbreB&);

        /**
        *  Destructeur de classe
        * */
        ~ArbreB();
```

Déroulement du code :

Nous avons choisis de baser nos fonctions sur un arbre binaire de recherche.

Le **constructeur** se base sur la méthode `ajout()` détaillée plus bas qui est implémentée pour ajouter un sommet dans toutes les situations y compris quand

l'arbre est vide. On peut donc ajouter plusieurs sommets via un tableau ainsi que sa taille.

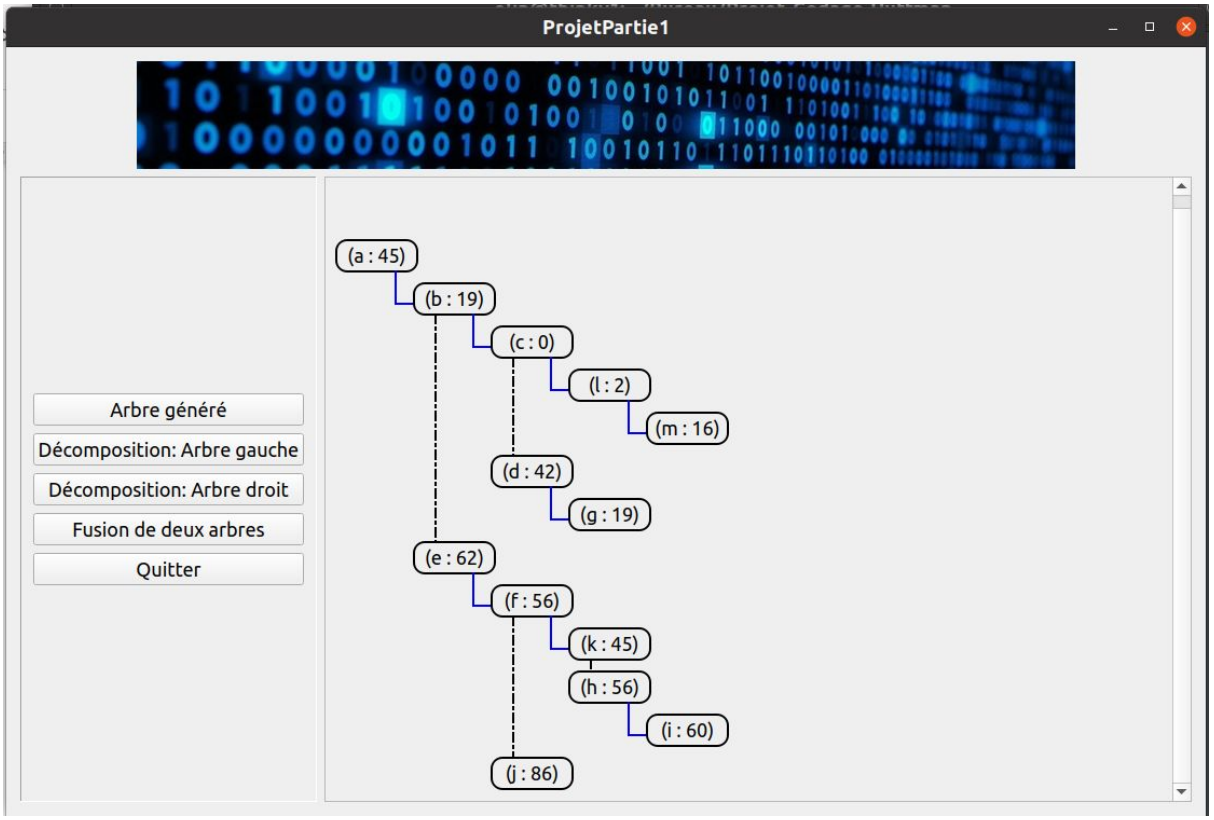
Cette méthode `ajout()` fonctionne récursivement et est surchargée trois fois : nous pouvons ajouter un sommet (ainsi que ses fils) à l'arbre courant, un ArbreB, ou un nœud avec sa @lettre et sa @fréquence.

La méthode d'affichage terminal se base sur un algorithme affichant l'arbre de la gauche vers la droite à raison d'un sommet par ligne. Deux fils d'un même sommets sont liés par un trait vertical noir. Un fils est lié à son père par un trait angulaire bleu.

La majeure partie des détails du code se trouvent en brief et commentaires de chacune des méthodes, aussi disponible sur la documentation générée par Doxygen.

Interface graphique :

L'interface graphique permis par Qt s'organise par l'affichage d'une fenêtre principale affichant un arbre binaire ainsi qu'un menu de bouton sur la gauche.



L'appui sur chacun des boutons reproduira quelques tests dont la décomposition et la fusion. Exécutez la version CLI en tapant `make cli` pour avoir la totalité des tests de méthodes.