Cahier des spécifications

Outil automatique de décryptage

Ce projet a été réalisé

par

Abdelaziz Ameurlain, Redha Dali, Mamadou Mouctar Barry, Allaye Afo Diallo, Rachid El Badaoui, Sofiane Hamad, Raphael Marouani, Linda Bedjaoui.

Chef du projet fonctionnel :
Linda Bedjaoui.

Sous la direction de :
Madame Leila Kloul

UVSQ
université PARIS-SACLAY

Table des matières

1 Introduction

L'application ODD (Outil automatique De Décryptage) a pour but d'être utilisée par n'importe quelle personne, qui voudrait chiffrer, déchiffrer et décrypter des messages confidentiels par l'utilisation des méthodes de Vigenère ou de Substitution. Toutes les fonctionnalités prévues lors de la phase de conception de l'application sont précisées dans ce document en indiquant l'implémentation de ces fonctionnalités dans l'application qui sera faite en langage C.

1.1 Objet du document

L'objet de ce document est de définir les spécifications de notre application « ODD» (Outil automatique De Décryptage) de façon détaillées. Les spécifications ont pour but de décrire précisément :

— Les différents modules de l'application ainsi que leurs fonctionnalités.
— Les types manipulés, leurs buts et leurs principes de fonctionnement.
— Les packages créés ainsi que l'explication de chacune de leurs fonctions et des bibliothèques utilisées.
— Explication sur le lien entre l'organigramme du cahier des charges et les spécifications.

1.2 Résumé des principales spécifications

Chiffrement. h :

- char* ChiffrementSubstitution(char* messageEntree, char* Cle)
- char* ChiffrementVigenere(char* messageEntree, char* Cle)

```
#ifndef CHIFFREMENT_H
#define CHIFFREMENT_H
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream>
// pour la manipulation des fichiers qui contiendront les textes en clair et chiffrés
#include <string.h>
// gerer la clef de chiffrement qui sera un attribut de type string

char* ChiffrementSubstitution(char* messageEntree, char* cle);

char* ChiffrementVigenere(char* messageEntree, char* cle);

#endif // CHIFFREMENT_H
```

Dechiffrement.h :

- char* dechiffrementSubstitution(char* messageEntree, char* Cle).
- char* dechiffrementVigenere(char* messageEntree, char* Cle).

```
#ifndef DECHIFFREMENT_H
#define DECHIFFREMENT_H

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream>
// pour la manipulation des fichiers qui contiendront les textes en clair et chiffrés
#include <string.h>
// gerer la clef de chiffrement qui sera un attribut de type string

char* DechiffrementSubstitution(char* messageEntree, char* cle);

char* DechiffrementVigenere(char* messageEntree, char* cle);

#endif // DECHIFFREMENT_H
```

Decryptage.h :

- freqlettre* analyseFrequence(char* messageEntree).
- char* decryptageSubstitution(char* messageEntree).
- int chercherOccurrence(char* messageEntree, char* sequence, int tailleSeq, int debut).
- void ajouterEcart(int ecart, int* taillesPossibles, int tailleProbable).
- int Kasiski(char* messageEntree).
- int longueurCleIC(char* messageEntree).
- char* rechercherCle(char* messageEntree, int tailleCle).
- char* decryptageVigenere(char* messageEntree).
- References.txt

```
#ifndef DECRYPTAGE_H
#define DECRYPTAGE_H

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fstream>
#include <string.h>


freqlettre* analyseFrequence(char* messageEntree);

char* decryptageSubstitution(char* messageEntree);

int chercherOccurrence(char* messageEntree, char* sequence, int tailleSeq, int debut);

void ajouterEcart(int ecart, int* taillesPossibles, int tailleProbable);

int Kasiski(char* messageEntree);

int longueurCleIC(char* messageEntree);

char* rechercherCle(char* messageEntree, int tailleCle);

char* decryptageVigenere(char* messageEntree);


#endif // DECRYPTAGE_H
```

Interface.h :

- gchar *ni_espace_ni_accent(gchar* texte).
- void fenetre_principale(int argc, char** argv).
- void cb_chiffrement(GtkWidget *pwidget, gpointer user_data).
- void cb_afficher_chif_substitution(GtkWidget *pwidget, gpointer user_data).
- void cb_afficher_chif_vigenere(GtkWidget *pwidget, gpointer user_data).
- void cb_dechiffrement(GtkWidget *pwidget, gpointer user_data).
- void cb_afficher_dechif_substitution(GtkWidget *pwidget, gpointer user_data).
- void cb_afficher_dechif_vigenere(GtkWidget *pwidget, gpointer user_data).
- void cb_decryptage(GtkWidget *pwidget, gpointer user_data).
- void cb_afficher_decrypt_substitution(GtkWidget *pwidget, gpointer user_data).
- void cb_afficher_decrypt_vigenere(GtkWidget *pwidget, gpointer user_data).
- void cb_analyse_frequence(GtkWidget *pwidget, gpointer user_data).
- void cb_afficher_frequence(GtkWidget *pwidget, gpointer user_data).
- void cb_coffre_fort(GtkWidget *pwidget, gpointer user_data).
- void cb_nouveau_coffre(GtkWidget *pwidget, gpointer user_data).
- void cb_mes_coffres(GtkWidget *pwidget, gpointer user_data).

```c
#include <stdlib.h>
#include <stdio.h>
#include <gtk/gtk.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include "Chiffrement.h"
#include "Dechiffrement.h"
#include "Decryptage.h"

gchar *ni_espace_ni_accent(gchar* texte);

void fenetre_principale(int argc, char** argv);

void cb_chiffrement(GtkWidget *pwidget, gpointer user_data);

void cb_afficher_chif_substitution(GtkWidget *pwidget, gpointer user_data);

void cb_afficher_chif_vigenere(GtkWidget *pwidget, gpointer user_data);

void cb_dechiffrement(GtkWidget *pwidget, gpointer user_data);

void cb_afficher_dechif_substitution(GtkWidget *pwidget, gpointer user_data);

void cb_afficher_dechif_vigenere(GtkWidget *pwidget, gpointer user_data);

void cb_decryptage(GtkWidget *pwidget, gpointer user_data);

void cb_afficher_decrypt_substitution(GtkWidget *pwidget, gpointer user_data);

void cb_afficher_decrypt_vigenere(GtkWidget *pwidget, gpointer user_data);

void cb_analyse_frequence(GtkWidget *pwidget, gpointer user_data);

void cb_afficher_frequence(GtkWidget *pwidget, gpointer user_data);

void cb_coffre_fort(GtkWidget *pwidget, gpointer user_data);

void cb_nouveau_coffre(GtkWidget *pwidget, gpointer user_data);

void cb_mes_coffres(GtkWidget *pwidget, gpointer user_data);
```

2 Glossaire

2.1 Données

Structure

La structure d'une donnée est un ensemble d'attributs qui définit son comportement. Ces principaux attributs sont :

1. L'intitulé.
2. Le type (int,float,char...).
3. Le format.

Valeurs

La valeur d'une donnée est un entier qui définira les longueurs de nos chaînes de caractères ou la fréquence d'apparition des lettres de l'alphabet par exemple.

Chaines de caractères

Les chaînes de caractères correspondront aux différents textes en entrée qu'on manipulera (texte en clair, texte chiffré, clé.). Ces derniers seront convertis tout en majuscule et sans espace au niveau de l'interface graphique avant de les envoyer aux modules concernés. Pour les textes en sortie seuls les textes en clair n'auront pas cette contrainte (ils seront affichés avec des espaces et la capitalisation.).

Tableaux

Les tableaux (qui peuvent avoir différentes dimensions) stockeront des valeurs et/ou caractères.

2.2 Bibliothéques

Dans l'implémentation de l'application, les différents fichiers d'en-tête et les bibliothèques qui seront utilisées dans ces derniers sont décrites ci dessous :

"Chiffrement.h" : Un fichier d'en-tête contenant la déclaration de toutes les fonctions et structures pour implémenter les différents algorithmes du module Chiffrement (Vigenère et par substitution).

"Dechiffrement.h" : Un fichier d'en-tête contenant la déclaration de toutes les fonctions et structures pour implémenter les différents algorithmes du module Dechiffrement (Vigenère et par substitution).

"Decryptage.h" : Un fichier d'en-tête contenant la déclaration de toutes les fonctions et structures pour implémenter les différents algorithmes du module Decryptage (Vigenère et par substitution).

"Interface.h" : Un fichier d'en-tête contenant la déclaration de toutes les fonctions et structures pour implémenter les algorithmes de l'interface graphique avec l'outil GTK.

<**stdio.h**> **:** Un fichier d'en-tête de la bibliothèque standard d'entrées/sorties du langage C. On y trouve des fonctions comme *fgetc* ou *fputc* qui représentent respectivement les fonctions de lecture et d'écriture dans des flux, ou encore la fonction *printf* pour l'écriture de données formatées qui sont indispensables lors de l'implémentation.

<**stdlib.h**> **:** Un fichier d'en-tête de la bibliothèque standard du langage C, utile pour l'exécution des diverses opérations dont la conversion, la génération de nombres pseudo-aléatoires, le contrôle de processus, la recherche, le tri, ou encore l'allocation de mémoire avec des fonctions comme malloc et free.

<**string.h**> **:** Un fichier d'en-tête de la bibliothèque standard du langage C, regroupant l'essentiel des fonctions nécessaires à la gestion des chaînes de caractères. Par exemple *strlen* qui permet de récupérer la longueur d'une chaîne de caractère ou *strcat* pour concaténer deux chaînes de caractères.

<**unistd.h**> **:** Un fichier d'en-tête qui permet d'accéder à l'interface de programmation des systèmes Unix, on l'utilisera principalement pour les fonctions d'appel système comme *pipe()*, *fort()* ou des fonctions d'entrées/sorties comme *read, write, close.*

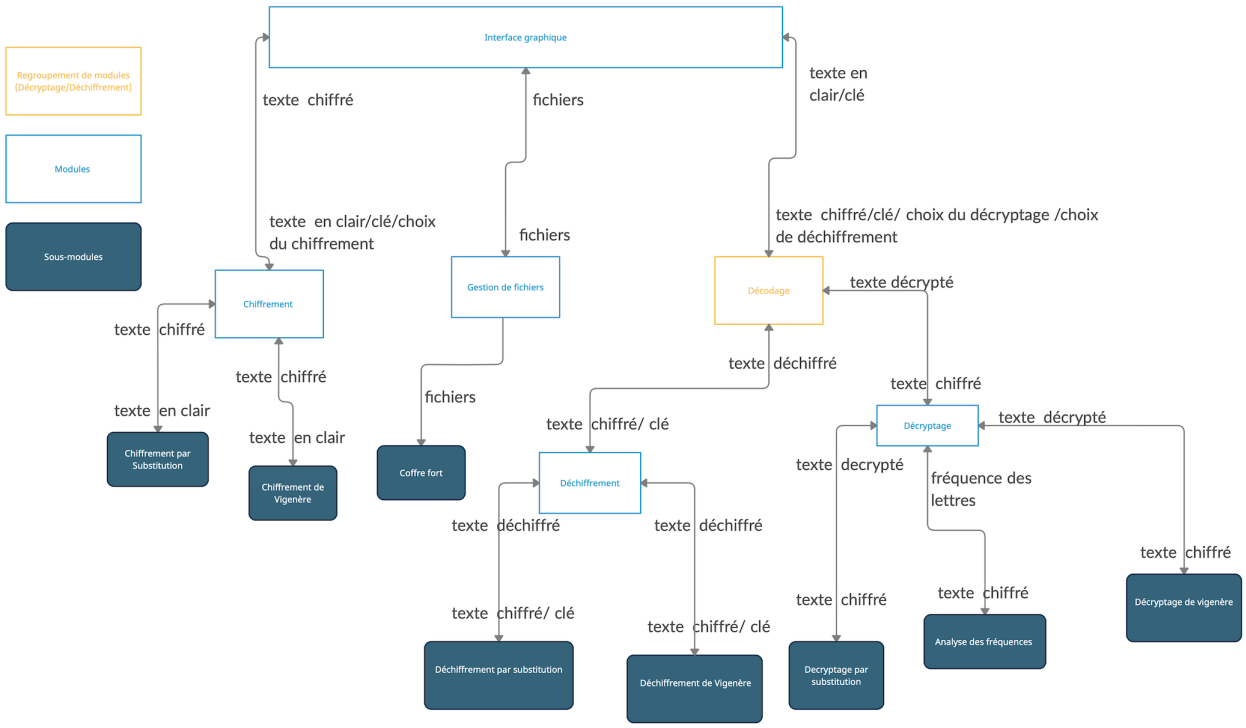
<**sys/types.h**> **:** On retrouve dans cette bibliothèque les définitions de types standard et de temps pour les systèmes Unix dans la gestion de fichiers. Par exemple, on aura besoin des types de données comme $MODE\_T$ qui permet d'indiquer les attributs d'un fichier.

<**sys/stat.h**> **:** Cette bibliothèque permet d'accéder aux caractéristiques d'un fichier. On y retrouve les fonctions pour créer, modifier, vérifier l'état d'un fichier ou d'un répertoire. Par exemple *mkdir* pour créer un répertoire ou *fstat* qui permet de demander l'état d'un fichier.

<**fcntl.h**> **:** Comme son nom l'indique, **«file control options»** ce fichier d'en-tête contient les fonctions comme *fcntl()* nécessaires à la manipulation des descripteurs de fichier. L'inclusion de l'en-tête <**fcntl.h**> peut également rendre visibles tous les symboles de <**sys/stat.h**> et <**unistd.h**>.

<**gtk/gtk.h**> **:** Un fichier contenant l'ensemble des fonctions nécessaires permettant l'implémentation de l'interface graphique avec l'outil GTK.

2.3 Précision sur l'organigramme



Les principaux changements effectués sur l'organigramme, ont été envisagés suite à notre réorientation vers le langage C. Les modules Chiffrement, Déchiffrement et Décryptage ne manipulent plus aucun fichier, ils échangent des données (chaînes de caractères, fréquences) avec l'interface graphique uniquement. Les liens reliant Chiffrement et Décodage à Gestion des fichiers (mémoire) ont par conséquent été supprimés. Pour la sauvegarde des données elle s'effectuera au niveau de l'interface graphique si l'utilisateur souhaite sauvegarder un message dans un fichier.

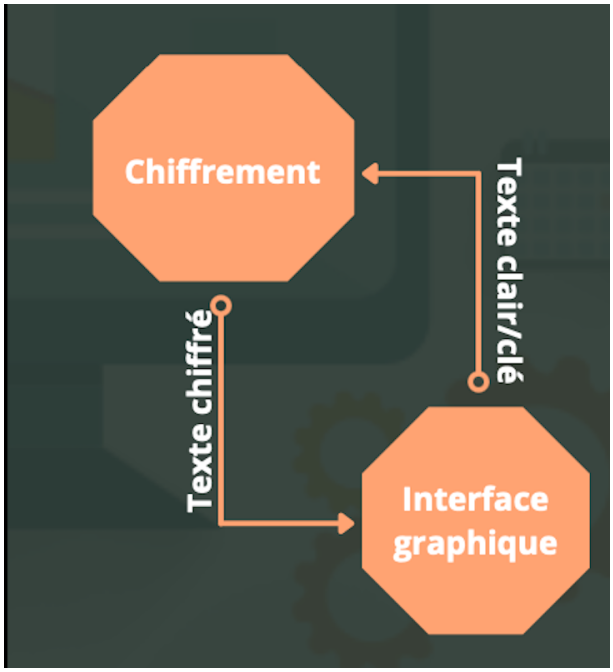
3 Explication du choix du langage

Notre application sera basée sur une programmation procédurale, un chiffrement ne pouvant pas être considéré comme une instance. Le C est un langage dit bas niveau, c'est-à-dire qu'il peut effectuer des opérations qui sont très proches du langage machine ce qui permet d'avoir des programmes plus rapides. De plus, c'est un langage qui peut s'exécuter sur toutes les plateformes (Linux, MacOS, Windows), ce qui est avantageux pour notre application. Nous avons également choisi une bibliothèque graphique adaptée au langage C (GTK+). Cette dernière nous permet de gérer efficacement la navigation entre les différentes fenêtres, ainsi que nos accès mémoires pour la sauvegarde et le chargement de nos fichiers.

4 Explication détaillée des spécifications

Ce dossier de spécifications sera applicable pendant la phase de développement de l'application « ODD». Le fonctionnement de l'application sera conforme aux éléments exactes présents dans ce dossier.

4.1 Module Chiffrement



Lien entre les modules chiffrement et interface graphique :

L'interface graphique enverra le texte en clair ainsi que la clé de chiffrement en argument en appellant la fonction ChiffrementSubstitution ou ChiffrementVigenere selon le choix de l'utilisateur. L'interface graphique affichera ensuite à l'utilisateur le résultat du chiffrement retourné par la fonction appellée.

Chiffrement par substitution

char* ChiffrementSubstitution(char* messageEntree, char* Cle) :

```
ChiffrementSubstitution(messageEntree,cle)
Entree : messageEntree /*texte en clair */ , cle /*la clé de chiffrement par substitution*/
Sortie : le message chiffre
DEBUT_ALGORITHME
POUR i ALLANT_DE 0 A 25
DEBUT_POUR
    tab[i] <-- 'A' + i; /* Intialiser le tableau de A à Z */
FIN_POUR
POUR i ALLANT_DE 0 A messageEntree.length
DEBUT_POUR
    POUR j ALLANT_DE 0 A 25
    DEBUT_POUR
    SI tab[j] == messageEntree[i]
    DEBUT_SI
       messageEntree[i] PREND_LA_VALEUR cle[j]
    FIN_SI
       messageSortie[i] PREND_LA_VALEUR messageEntree[i]
    FIN_POUR
FIN_POUR
RETOUNER messageSortie
FIN_ALGORITHME
```

Cette fonction utilisera l'algorithme de chiffrement par substitution mono-alphabetique pour chiffrer un message en clair avec la clé entrée par l'utilisateur et renverra le message chiffré. Elle prendra en argument : le texte en clair et la clé de chiffrement.

Le déroulement de la fonction :
- Lecture caractère par caractère de la Cle (les 26 lettres de l'alphabet entrées dans l'ordre choisi par l'utilisateur).
- Vérification de la taille de la clé (égale à 26) et de l'absence de doublons.
- Lecture caractère par caractère du messageEntree (le texte à chiffrer entré par l'utilisateur).
- Identification du caractère lu à l'aide du code ASCII.

— Substitution du caractère par la lettre correspondante dans la clé.
— Initialisation du char* messageSortie à " " (chaîne vide).
— Construction de la chaîne de caractère messageSortie (le texte chiffré) en ajoutant la lettre substituée.
— Répétition de l'opération jusqu'à la fin du messageEntree.

Chiffrement de Vigenère

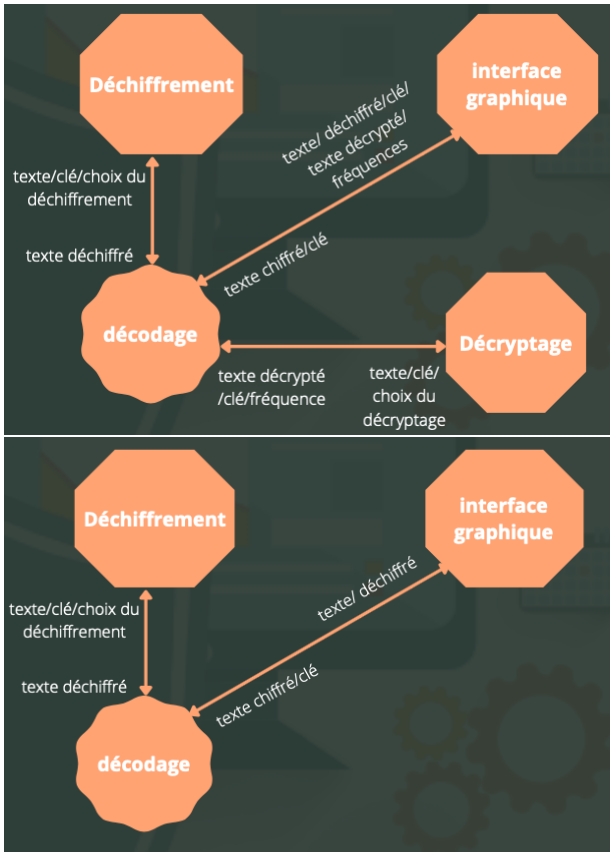
char* ChiffrementVigenere(char*messageEntree, char* Cle) :

```
ChiffrementVigenere(messageEntree,cle)
VARIABLES
    messageEntree EST_DU_TYPE CHAINE
    i EST_DU_TYPE NOMBRE
    code_lettre EST_DU_TYPE NOMBRE
    cle EST_DU_TYPE CHAINE
    longueur_message EST_DU_TYPE NOMBRE
    longueur_cle EST_DU_TYPE NOMBRE
    decalage EST_DU_TYPE NOMBRE
    lettre EST_DU_TYPE CHAINE
DEBUT_ALGORITHME
    LIRE messageEntree
    LIRE cle
    longueur_message PREND_LA_VALEUR messageEntree.length
    longueur_cle PREND_LA_VALEUR cle.length
    POUR i ALLANT_DE 1 A longueur_message
        DEBUT_POUR
        code_lettre PREND_LA_VALEUR messageEntree.charCodeAt(i-1)-65
        decalage PREND_LA_VALEUR cle.charCodeAt((i-1)%longueur_cle)-65
        lettre PREND_LA_VALEUR String.fromCharCode(65+(code_lettre+decalage)%26)
    AFFICHER lettre
    FIN_POUR
FIN_ALGORITHME
```

Cette fonction utilisera l'algorithme de chiffrement de Vigenère poly-alphabetique pour chiffrer un message en clair avec la clé entrée par l'utilisateur et renverra le message chiffré. Elle prendra en argument : le texte en clair et la clé de chiffrement.

Le déroulement de la fonction :
— Lecture caractère par caractère de la variable Cle.
— Vérification de la taille de la clé (inferieure ou égale à la taille du message en entrée).
— Lecture caractère par caractère du messageEntree.
— À l'aide d'une opération modulo entre la position du caractère étudiée dans le message en clair et de la taille de la clé (key), nous déterminons la lettre de la clé qui sera utilisée afin de chiffrer l'occurence de cette lettre.
— Addition du code ASCII de la lettre étudiée avec celui de la lettre de la clé correspondante en soustrayant deux fois le code ASCII de la lettre A (65) puis conversion en modulo 26 de la somme calculée.
— Addition de l'indice trouvé et du code ASCII de la lettre A (65) permettant d'obtenir celui de la lettre substituant l'occurence du caractère étudié.
— Initialisation du char* messageSortie à " " (chaîne vide).
— Construction de la chaîne de caractère messageSortie (le texte chiffré) en ajoutant la lettre substituée.
— Répétition de l'opération jusqu'à la fin du messageEntree.

4.2 Module Déchiffrement



Précision sur le regrouprement de modules (Décodage) :

Décodage ne constitue pas un module à part entière mais simplement un regroupement de modules englobant les modules Déchiffrement et Décryptage, dans le but de rendre l'organigramme le plus clair possible visuellement.

Lien entre les modules Déchiffrement et Interface graphique :

L'interface graphique enverra le texte chiffré ainsi que la clé de chiffrement en argument en appellant la fonction DechiffrementSubstitution ou DechiffrementVigenere selon le choix de l'utilisateur. L'interface graphique affichera ensuite à l'utilisateur le résultat du déchiffrement retourné par la fonction appellée.

Déchiffrement par substitution

char* DechiffrementSubstitution(char* messageEntree, char* Cle) :

```
DechiffrementSubstitution(messageEntree,cle)
Entree : messageEntree /*texte chiffre */ , cle /*la clé de chiffrement par substitution*/
Sortie : le message en clair
DEBUT_ALGORITHME
POUR i ALLANT_DE 0 A 25
DEBUT_POUR
    tab[i] <-- 'A' + i; /* Intialiser le tableau de A à Z */
FIN_POUR
POUR i ALLANT_DE 0 A messageEntree.length
    POUR j ALLANT_DE 0 A 25
DEBUT_POUR
SI tab[j] == messageEntree[i]
    DEBUT_SI
       messageEntree[i] PREND_LA_VALEUR cle[j]
    FIN_SI
       messageSortie[i] PREND_LA_VALEUR messageEntree[i]
    FIN_POUR
FIN_POUR
RETOUNER messageSortie
FIN_ALGORITHME
```

Cette fonction utilisera l'algorithme de déchiffrement par substitution mono-alphabetique pour dechiffrer un message crypté avec la clé entrée par l'utilisateur et renverra le message en clair. Elle prendra en argument : le texte chiffré et la clé de déchiffrement.

Le déroulement de la fonction :
- Lecture caractère par caractère de la Cle (les 26 lettres de l'alphabet entrées dans l'ordre choisi par l'utilisateur).
- Vérification de la taille de la clé (égale à 26) et de l'absence de doublons.
- Lecture caractère par caractère du messageEntree (le texte à déchiffrer entré par l'utilisateur).
- Identification du caractère lu à l'aide du code ASCII.
- Substitution du caractère par la lettre correspondante dans la clé.
- Initialisation du char* messageSortie à " " (chaîne vide).
- Construction de la chaîne de caractère messageSortie (le texte en clair) en ajoutant la lettre substituée.
- Répétition de l'opération jusqu'à la fin de messageEntree.

Déchiffrement de Vigenère

char* DechiffrementVigenere(char* messageEntree, char* Cle) :

```
DechiffrementVigenere(messageEntree,cle)
VARIABLES
    messageEntree EST_DU_TYPE CHAINE
    i EST_DU_TYPE NOMBRE
    code_lettre EST_DU_TYPE NOMBRE
    cle EST_DU_TYPE CHAINE
    longueur_message EST_DU_TYPE NOMBRE
    longueur_cle EST_DU_TYPE NOMBRE
    decalage EST_DU_TYPE NOMBRE
    lettre EST_DU_TYPE CHAINE
DEBUT_ALGORITHME
    LIRE messageEntree
    LIRE cle
    longueur_message PREND_LA_VALEUR messageEntree.length
    longueur_cle PREND_LA_VALEUR cle.length
    POUR i ALLANT_DE 1 A longueur_message
        DEBUT_POUR
        code_lettre PREND_LA_VALEUR messageEntree.charCodeAt(i-1)-65
        decalage PREND_LA_VALEUR cle.charCodeAt((i-1)%longueur_cle)-65
        SI (code_lettre - decalage >= 0) ALORS
        DEBUT_SI
            lettre PREND_LA_VALEUR String.fromCharCode(65+(code_lettre+decalage)%26)
        FIN_SI
        SINON
        DEBUT_SINON
            lettre PREND_LA_VALEUR String.fromCharCode(65+(code_lettre+decalage)%26+26)
        FIN_SINON
    AFFICHER lettre
    FIN_POUR
FIN_ALGORITHME
```

Cette fonction utilisera l'algorithme de déchiffrement de Vigenère poly-alphabetique pour déchiffrer un message crypté avec la clé entrée par l'utilisateur et renverra le message en clair. Elle prendra en argument : le texte chiffré et la clé de déchiffrement.

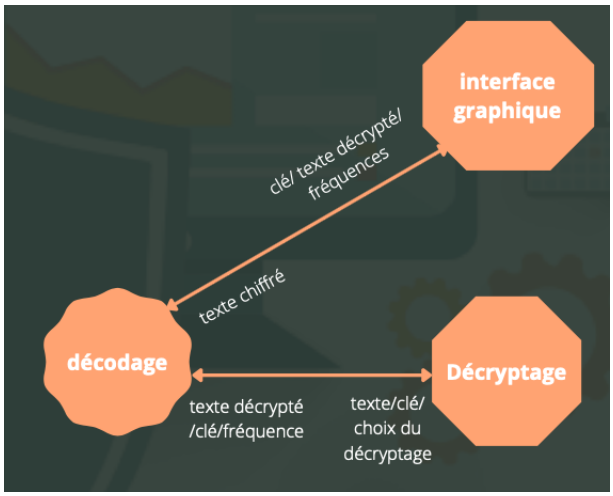
Le déroulement de la fonction :
- Lecture caractère par caractère de la Cle.
- Vérification de la taille de la clé (inferieure ou égale à la taille du message en entrée).
- Lecture caractère par caractère du messageEntree.
- À l'aide d'une opération modulo entre la position du caractère étudiée dans le message en clair et de la taille de la clé (key), nous déterminons la lettre de la clé qui sera utilisée afin de déchiffrer l'occurence de cette lettre.
- Soustraction du code ASCII de la lettre étudiée avec celui de la lettre de la clé correspondante puis faire le modulo 26 de la somme calculée.
- Addition de l'indice trouvé et du code ASCII de la lettre A (65) permettant d'obtenir celui de la lettre substituant l'occurence du caractère étudié.

— Initialisation du char* messageSortie à " " (chaîne vide).
— Construction de la chaîne de caractère messageSortie (le texte en clair) en ajoutant la lettre substituée.
— Répétition de l'opération jusqu'à la fin de messageEntree.

4.3 Module Décryptage

Lien entre les modules Décryptage et Interface graphique :

En cas de décryptage désiré par l'utilisateur et en fonction de son choix, l'interface graphique enverra le texte chiffré au module Décryptage en argument d'une des deux fonctions DecryptageSubstitution(texte chiffré) et DecryptageVigenere(texte chiffré) L'interface graphique pourra également envoyer un texte quelconque en argument de la fonction freqlettre* analyseFrequence(texte) si l'utilisateur souhaite simplement effectuer une analyse de fréquence. L'interface graphique affiche ensuite à l'utilisateur le résultat du décryptage ou de l'analyse de fréquence retourné par la fonction appellée.



References.txt :

```
Analyse des fréquences de la langue française:

E A I S T N R U L O D M P C V Q G B F J H Z X Y K W

Lettre  Pourcentage
A       9,42
B       1,02
C       2,64
D       3,39
E       15,87
F       0,95
G       1,04
H       0,77
I       8,41
J       0,89
K       0,00
L       5,34
M       3,24
N       7,15
O       5,14
P       2,86
Q       1,06
R       6,46
S       7,90
T       7,26
U       6,24
V       2,15
W       0,00
X       0,30
Y       0,24
Z       0,32
```

Description : Ce fichier contient l'analyse des fréquences de la langue française récupérée via internet.
https ://fr.wikipedia.org/wiki/Analyse_fréquentielle

freqlettre* analyseFrequence(char* messageEntree) :

```
analyseFrequence(messageEntree):tab
Entrée   : message en entrée
Sortie   : tableau de frequences
variable : tab : tableau de 26 cases
tab <-- {0}
POUR i ALLANT_DE 1 A messageEntree.length
DEBUT_POUR
    tab[messageEntree[i] -'A'] <-- tab[messageEntree[i]-'A'] + 1/messageEntree.length;
FIN_POUR
RETOUNER tab
FIN_ALGORITHME
```

Cette fonction utilisera l'algorithme d'analyse des fréquences ainsi qu'une structure nommée freqlettre pour calculer la fréquences d'apparition des lettres de l'alphabet dans un texte. Elle prendra en argument le message sur lequel on veut appliquer l'algorithme.

Les principaux attributs de la structure freqlettre.

struct *freqlettre*
{
    char* lettre ; - - la lettre ou le couple de lettres dont on veut calculer la fréquence.
    float freq ; - - la fréquence d'apparition de la lettre ou du couple de lettres.
};

Le déroulement de la fonction :
— Lecture caractère par caractère du messageEntree.
— Déclaration dynamique d'un tableau à 26 cases de type freqlettre.
— Initialisation du tableau avec les lettres de l'alphabet (A/Z) et les frequences à 0.
— Lecture caractère par caractère du message en entrée.
— Identification du caractère lu à l'aide du code ASCII.
— Soustraction du code ASCII de la lettre étudiée avec celui de la lettre A(65) pour obtenir l'indice de la case du tableau.
— Faire l'addition de $(1 \div tailledutexte)$ à la fréquence de la case correspondante.
— À la fin de la boucle retourner le tableau.

Décryptage par substitution

char* DecryptageSubstitution(char* messageEntree) :

```
DecryptageSubstitution(messageEntree)
Entree : messageEntree /*texte chiffre */
Sortie : messageSortie /*le message en clair*/
variables : tab_ref[26] : tableau de frequences de la langue francaise
            tab_freq[26] : tableau de frequences du messageEntree

DEBUT_ALGORITHME

tab_ref[26] <-- 'E' ,'A' ,'I', 'S', 'T' ,'N', 'R' ,'U' ,'L' ,'O', 'D' ,'M' ,'P',
                'C' ,'V', 'Q', 'G' ,'B', 'F', 'J', 'H' ,'Z' ,'X' ,'Y', 'K', 'W';
POUR i ALLANT_DE 0 A 25
DEBUT_POUR
    tab_freq[i] <-- max(analyseFrequence(messageEntree)); /* la fonction max retourne la lettre la
                                                             plus frequente dans le messageEntree
                                                             a partir de l'indice du tableau et mets
                                                             la valeur correspondante a -1 pour pas
                                                             stocker la meme lettre plusieurs fois */
FIN_POUR
POUR i ALLANT_DE 0 A messageEntree.length
DEBUT_POUR
    POUR j ALLANT_DE 0 A 25
    DEBUT_POUR
    SI tab_freq[j] == messageEntree[i]
    DEBUT_SI
       messageEntree[i] PREND_LA_VALEUR tab_ref[j] /* On remplace chaque lettre par sa correspondante
                                                      ( si X est le plus frequant dans messageEntree il
                                                      sera remplacé par E qui est le plus frequant dans
                                                      la langue francaise ainsi de suite */
    FIN_SI
       messageSortie[i] PREND_LA_VALEUR messageEntree[i]
    FIN_POUR
FIN_POUR
RETOUNER messageSortie //retourne texte ne clair
FIN_ALGORITHME
```

Cette fonction utilisera l'algorithme de décryptage par substitution mono-alphabetique pour décrypter un message chiffré entré par l'utilisateur et retournera le message en clair.

Le déroulement de la fonction :

— Déclarer un tableau (tabRef) de 26 cases de type freqlettre.
— Initialiser le tableau (tabRef) de façon décroissante en se référant aux données du fichier References.txt.
— Lecture caractère par caractère du messageEntree.
— Appeler la fonction freqlettre* analyseFrequence(messageEntree) afin de faire l'analyse des fréquences des lettres du message chiffré et trier le tableau (tabFreq) en décroissant.
— Récuperer la lettre de la i ème case (i allant de 0 à 25) du tableau (tabFreq).
— Rechercher dans le message chiffré toute les lettres identiques à la lettre récupérée et les remplacer par la lettre du tableau (tabRef) à la position i (la même position de la lettre récupérée).
— Initialisation du char* messageSortie à " " (chaîne vide).
— Construction de la chaîne de caractère messageSortie (le texte en clair) en ajoutant la lettre remplacée (la lettre du tableau tabRef).
— Répétition de l'opération jusqu'à la fin du messageEntree.

Décryptage de Vigenère

Le décryptage de Vigenère utilisera plusieurs fonctions décrites ci-dessous :

int ChercherOccurrence(char* messageEntree, char* sequence, int tailleSeq, int debut)

```
chercherOccurrence(chaine[],taille,debut)
Entree : chaine /* message en entree /, taille / la taille de la chaine /, debut / indice de départ */
Sortie : le nombre d occurence de chaque sequence (chaine) donnee en entrée
DEBUT_ALGORITHME
POUR i ALLANT_DE debut A tailleMessage - taille
DEBUT_POUR
iTab <-- i
iChaine <-- 0
    TANT_QUE iTab < tailleMessage - taille + 1 ET chaine[iChaine] == message[iTab]
    DEBUT_TANT_QUE
        iTab <-- iTab + 1
        iChaine <-- iChaine + 1
    FIN_TANT_QUE
SI iChaine == taille
DEBUT_SI
    RETOURNER i
FIN_SI
FIN_POUR
FIN_ALGORITHME
```

Cette fonction recherche dans le message chiffré une séquence identique à celle étudiée et si jamais une telle séquence est trouvée, retourne l'écart en nombre de caractères séparant les deux séquences identiques. Si aucune séquence identique n'est trouvée, la fonction retourne -1.

Le déroulement de la fonction :
— Lecture du messageEntree en partant du caractère à la position debut.
— Comparaison de la séquence étudiée avec chaque séquence lue.
— Calcul de la distance séparant les deux séquences si elles sont identiques et la retourner.
— Dans le cas où aucune séquence identique n'est trouvée retourner -1.

void AjouterEcart(int ecart, int* taillesPossibles, int tailleProbable)

```
ajouterEcart(ecart,taillesPossibles,tailleProbable)

Entrée   : ecart, tableau d entier (taillesPossibles)
Sortie   : tailleProbable

POUR i ALLANT_DE 2 A ecart
    DEBUT_POUR
    SI (ecart % i == 0) ALORS
        DEBUT_SI
            ++taillesPossibles[i];
        SI (taillesPossibles[i] > taillesPossibles[tailleProbable]) ALORS
            DEBUT_SI
            tailleProbable = i;
        FIN_SI
     FIN_SI
FIN_POUR
```

Cette fonction enregistre les écarts entre les séquences du message chiffré qui se répètent, les stockent dans un tableau nommé taillesPossibles et note (tailleProbable) l'indice de la case du tableau correspondant à la taille de clé soupsonnée être celle que l'on cherche.

Le déroulement de la fonction :
— Pour chaque valeur entre 2 et l'écart étudié, vérifier que la valeur soit un diviseur de l'écart à l'aide d'une opération modulo.
— Si la valeur est bien un diviseur de l'écart, alors incrémenter la case correspondante du tableau taillesPossibles.
— Comparaison de la valeur de la case du tableau venant d'être incrémentée avec la valeur de la case du tableau correspondant à l'indice tailleProbable. Si cette dernière est inférieure alors la valeur de tailleProbable est remplacée par le nouvel indice du tableau.

int Kasiski(char* messageEntree) :

```
Kasiki(messageEntree)
Entree : message cbiffré
Sortie : longueur de la cle (ou le diviseur commun minimum)
DEBUT_ALGORITHME
   taille_sequence <-- 3
   bool trouve <-- true // bool trouve = true
   taille_message <-- messageEntree.length
   TANT_QUE(trouve == true ) FAIRE
   DEBUT_TANT_QUE
      trouve <-- false
      POUR debut ALLANT DE 0 A taille_message - taille_sequence  + 1
      DEBUT_POUR
          chaine[taille_sequence + 1]
         POUR i ALLANT DE 0 A taille_sequence
         DEBUT_POUR
            chaine[i] <-- message[debut + i];
         FIN_POUR
         chaine[i] <-- '\0'
         occurrence <-- chercherOccurrence(chaine, taille_sequence, debut + taille_sequence)
         SI (occurrence != -1 ) ALORS
         DEBUT_SI
            ecart <-- occurrence - iDebut;
            trouve <-- true;
            ajouterEcart(ecart)
         FIN_SI
      FIN_POUR
      taille_sequence <-- taille_sequence+1
   FIN_TANT_QUE
   RETOUNER PGCDEcart()
```

Cette fonction utilisera l'algorithme de Kasiski pour déterminer la taille de la clé utilisée pour chiffrer le messageEntree et la retournera (ou retournera son plus petit diviseur). Elle prendra en argument le texte chiffré.

Le déroulement de la fonction :
— Pour chaque séquence de 3 caractères du message contenu dans le fichier d'entrée, la stocker dans un tableau de char puis appeler la fonction chercherOccurence qui renverra l'écart en nombre de caractères avec une séquence identique si elle est trouvée. Dans le cas contraire elle renverra -1.
— Si un écart est trouvé, appeler la fonction ajouterEcart qui tiendra à jour le tableau de int tailles-Possibles contenant les diviseurs des écarts et la valeur tailleProbable étant l'indice de la case du tableau taillesPossibles contenant la valeur la plus grande.
— Une fois toutes les séquences de 3 caractères étudiées, passer aux séquences de 4 caractères puis 5 etc... Jusqu'à la taille du message.
— Une fois toutes les séquences étudiées, retourner la variable tailleProbable qui représente la taille probable de la clé.

int longueurCleIC(char* messageEntree) :

```
longueurCleIC(messageEntree)
DEBUT_ALGORITHME
    LIRE messageEntree  tab[0] PREND_LA_VALEUR -1
    POUR i ALLANT_DE 1 A messageEntree.length
    DEBUT_POUR
        tampon PREND_LA_VALEUR ""   Ic PREND_LA_VALEUR 0
        POUR l ALLANT_DE 1 A tampon.length
            DEBUT_POUR
            a[l] PREND_LA_VALEUR 0
            FIN_POUR
        k PREND_LA_VALEUR  0
        TANT_QUE (k<= messageEntree.length) FAIRE
            DEBUT_TANT_QUE
            tampon PREND_LA_VALEUR tampon + messageEntree.substr(k,1) k PREND_LA_VALEUR  k+i
            FIN_TANT_QUE
        POUR j ALLANT_DE 1 A tampon.length
            DEBUT_POUR
            a[j] PREND_LA_VALEUR tampon.charCodeAt(j-1)
            FIN_POUR
        POUR j ALLANT_DE 65 A 90
         DEBUT_POUR
            N PREND_LA_VALEUR 0
            POUR m ALLANT_DE 1 A tempon.length
                DEBUT_POUR
                SI (a[m] == j) ALORS
                    DEBUT_SI
                    N PREND_LA_VALEUR  N+1
                    FIN_SI
                FIN_POUR
            Ic PREND_LA_VALEUR  ic+N(N-1)/(tempon.length(tampon.length-1))
          FIN_POUR
        tab[i] PREND_LA_VALEUR ic
    FIN_POUR
    indice PREND_LA_VALEUR -1  max PREND_LA_VALEUR -1
    POUR i ALLANT_DE 0 A messageEntree.length
    DEBUT_POUR
        SI max < tab[i] ALORS
            DEBUT_SI
            max PREND_LA_VALEUR tab[i]  indice PREND_LA_VALEUR i
        FIN_SI
FIN_POUR
RETOUNER indice
FIN_ALGORITHME
```

Cette fonction utilisera l'algorithme d'indice de coïncidence pour déterminer la taille de la clé utilisée pour chiffrer le messageEntree et retournera la valeur calculée.

Le déroulement de la fonction :
— Lecture caractére sur i (i prend la valeur 2 au départ) du messageEntree.
— Appliquer la formule d'indice de coïncidence.
— Répétition de l'opération jusqu'à la fin du messageEntree.
— Incrementer i de 1.
— Répétition de l'opération jusqu'à trouver l'indice le plus proche de 0.74.

Aprés avoir effectué les deux algortithmes (Kasiski et indice de coïncidence) il va falloir vérifier que les deux valeurs obtenues concordent. Il y a 3 options : - Les deux tailles trouvées sont strictement identiques, la suite du décryptage peut donc continuer. - Les deux tailles trouvées sont différentes mais celle trouvée via la méthode de Kasiski est un diviseur de celle calculée via l'indice de coïncidence, la suite du décryptage peut donc se poursuivre en prenant en compte la taille trouvée grâce à l'indice de coïncidence. - Les deux tailles trouvées n'ont complètement rien à voir, le décryptage s'arrête donc et un message d'erreur est renvoyé.

char* RechercherCle(char* messageEntree, int tailleCle) :

```
rechercherCle(char* messageEntree, int tailleCle)
Entree : messageEntree /*texte chiffrée */ , tailleCle /*longueur de la clé*/
Sortie : la cle en clair
Variables : tab /*tableau de chaîne de caractere */, cle /* chaîne de caractere */
            freq /*tableau contenant la féquence des lettres du message chiffré */
            max /* nombre entier */ , indice /* nombre entier */

DEBUT_ALGORITHME
cle PREND_LA_VALEUR ""

POUR i ALLANT_DE 0 A tailleCle - 1
DEBUT_POUR
    tab[i] PREND_LA_VALEUR ""   // initaliser les chaîne dans le tableau à null
FIN_POUR
POUR i ALLANT_DE 0 A messageEntree.length
DEBUT_POUR
    tab[i%tailleCle] += messageEntree[i]  // on separe le message en tailleCle lignes
FIN_POUR
POUR i ALLANT_DE 0 A tailleCle - 1
DEBUT_POUR
    freq += analyseFrequence(tab[i])
    max  PREND_LA_VALEUR 0
    indice PREND_LA_VALEUR 0
FIN_POUR
POUR j ALLANT_DE 0 A 25
DEBUT_POUR
        SI max < freq[j] ALORS
            DEBUT_SI
            max PREND_LA_VALEUR freq[j]
            indice PREND_LA_VALEUR j
        FIN_SI
FIN_POUR
cle += (indice + 22)%26 //indice est la position de la lettre la plus frequente
                        // 22 est le decalage pour trouver la clé

RETOUNER cle
FIN_ALGORITHME
```

Cette fonction a pour objectif de trouver les lettres qui constituent la clé. Elle prendra en argument : le texte chiffré et la taille de la clé.

Le déroulement de la fonction :

— Initialisation de la chîne de caractére Cle à " " (chaîne vide).
— Initialisation d'un tableau de caractére " " (vide) de taille tailleCle.
— Découpe du messageEntree en tailleCle lignes (si tailleCle = 6 on découpe le messageEntree en 6 lignes, la première ligne contiendra la 1ère lettre, la 7ème lettre puis la 13ème lettre etc... La deuxiéme ligne contiendra la 2ème lettre, la 8ème lettre, puis la 14ème lettre etc... Et ainsi de suite).
— Appel à la fonction analyseFrequence sur chaque bout du messageEntree obtenu précédemment.
— Constitution de la Cle avec les lettres trouvées aprés l'analyse de fréquence de chaque ligne.

char* DecryptageVigenere(char* messageEntree) :

```
decryptageVigenere(messageEntree)

DEBUT_ALGORITHME
i PREND_LA_VALEUR 0
j PREND_LA_VALEUR 0
tailleCle PREND_LA_VALEUR 0
cle PREND_LA_VALEUR ""
messageSortie PREND_LA_VALEUR ""

i <-- longueurCleIC(messageEntree)
j <-- Kasiki(messageEntree)

SI i == j OU (j/i == i) ALORS
    DEBUT_SI
       tailleCle PREND_LA_VALEUR j
    FIN_SI
SINON
    "Erreur dans la taille de la cle"
cle <-- rechercherCle(messageEntree,tailleCle)
messageSortie <-- dechiffrementVigenere(messageEntree, cle)

RETOUNER messageSortie
FIN_ALGORITHME
```

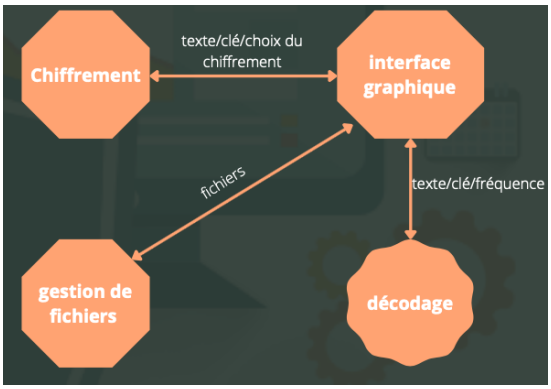
Cette fonction fera appel aux précedentes fonctions ainsi qu'a la fonction char* dechiffrementVigenere(char* messageEntree, char* Cle) pour décrypter le message chiffré. Elle prendra en argument : le texte chiffré et la taille et retournera le message en clair.

Le déroulement de la fonction :
— Appel aux fonctions Kasiki et longueurIcCle pour déterminer la taille de la clé.
— Vérification de la concordance entre les deux valeurs.
— Appel à la fonction rechercherCle pour constituer notre clé.
— Appel à la fonction dechiffrementVigenere pour décrypter notre messageEntree.

4.4 Module Interface graphique

Lien entre les modules Décryptage et Interface graphique :

L'interface graphique enverra le texte ou le texte et la clé au module sollicité en argument de la fonction représentant la fonctionnalité choisie par l'utilisateur. L'interface graphique affiche ensuite à l'utilisateur le résultat retourné par la fonction appellée.



gchar *ni_espace_ni_accent(gchar* texte) :
— Déclaration d'une variable char* qui contiendra le texte sans espace, sans accent et en majuscule.
— Lecture de la lettre.
— Vérification si la lettre lue est en majuscule, si oui, on la met dans la variable.
— Vérification si la lettre lue est en minuscule, si oui, on la met en majuscule dans la variable.
— Vérification si la lettre lue à un accent, si oui, on met sa majuscule sans accent correspondante dans la variable.
— on répète l'opération tant qu'on a pas finir de lire le texte.

void fenetre_principale(int argc, char argv) :**
La fenêtre principale qui contiendra le menu et les boutons chiffrement, déchiffrement, décryptage, analyse des fréquences, coffre-fort. Les boutons seront connectés respectivement aux fonctions cb$_chiffrement, cb_dechiffr$
Le menu contiendra chiffrement, déchiffrement et décryptage qui seront composés de substitution et Vigenère qui seront directement connectés à leur fenêtre respective.

void cb_chiffrement(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Création des espaces de saisie du texte et de la clé.
— Création des boutons substitution et vigenere.
— Connexion des boutons à leur fonction callback respective (cb_chiffrement_substitution et cb_chiffrement_vige
— À cette étape, il y a une circulation d'information de l'interface à chiffrement. On sait sur quel bouton l'utilisateur à cliquer, grâce à cette information, on appelle la bonne fonction de chiffrement.

void cb_afficher_chif_substitution(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Récupération du texte et de la clé.
— Appel de la fonction ni_espace_ni_accent sur le texte puis sur la clé.
— Appel de la fonction chiffrementSubstitution sur le texte sans espace obtenu précédemment et la clé. À cette étape, il y a une circulation l'information entre l'interface et chiffrement. L'interface envoie le texte clair et la clé puis chiffrement retourne le texte chiffré.
— Affichage du texte chiffré, demande de sauvegarde : si oui échange entre interface et gestion de fichier.

void cb_afficher_chif_vigenere(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Récupération du texte et de la clé.
— Appel de la fonction ni_espace_ni_accent sur le texte puis sur la clé.
— Appel de la fonction chiffrementSubstitution sur le texte sans espace obtenu précédemment et la clé. À cette étape, il y a une circulation l'information entre l'interface et chiffrement. L'interface envoie le texte clair et la clé puis chiffrement retourne le texte chiffré.
— Affichage du texte chiffré, demande de sauvegarde : si oui échange entre interface et gestion de fichier.

void cb_dechiffrement(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Récupération du texte et de la clé.
— Appel de la fonction ni_espace_ni_accent sur le texte puis sur la clé.
— Appel de la fonction chiffrementSubstitution sur le texte sans espace obtenu précédemment et la clé. À cette étape, il y a une circulation l'information entre l'interface et chiffrement. L'interface envoie le texte clair et la clé puis chiffrement retourne le texte chiffré.
— Affichage du texte chiffré, demande de sauvegarde : si oui échange entre interface et gestion de fichier.

void cb_afficher_dechif_substitution(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Récupération du texte et de la clé.
— Appel de la fonction ni_espace_ni_accent sur le texte puis sur la clé.
— Appel de la fonction dechiffrementSubstitution sur le texte sans espace obtenu précédemment et la clé. À cette étape, il y a une circulation l'information entre l'interface et chiffrement. L'interface envoie le texte chiffré et la clé puis déchiffrement retourne le texte déchiffré.

— Affichage du texte déchiffré, demande de sauvegarde : si oui échange entre interface et gestion de fichier.

void cb_afficher_dechif_vigenere(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre
— Récupération du texte et de la clé
— Appel de la fonction ni_espace_ni_accent sur le texte puis sur la clé
— Appel de la fonction dechiffrementSubstitution sur le texte sans espace obtenu précédemment et la clé. À cette étape, il y a une circulation l'information entre l'interface graphique et le déchiffrement.
— L'interface envoie le texte chiffré et la clé puis déchiffrement retourne le texte déchiffré.
— Affichage du texte déchiffré, demande de sauvegarde : si oui échange entre interface et gestion de fichier.

void cb_decryptage(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Création de l'espace de saisie du texte.
— Création des boutons substitution et vigenere
— Connexion des boutons à leur fonction callback respective (cb_decryptage_substitution et cb_decryptage_vige
— À cette étape, il y a une circulation d'information de l'interface à décryptage. On sait sur quel bouton l'utilisateur à cliquer, grâce à cette information, on appelle la bonne fonction de décryptage.

void cb_afficher_decrypt_substitution(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Récupération du texte.
— Appel de la fonction ni_espace_ni_accent sur le texte.
— Appel de la fonction decryptageSubstitution sur le texte sans espace obtenu précédemment. À cette étape, il y a une circulation l'information entre l'interface graphique et le décryptage. L'interface envoie le texte clair puis le décryptage retourne le texte décrypte à l'interface.
— Affichage du texte décrypté et de la clé potentielle, demande de sauvegarde : si oui échange entre interface et gestion de fichier.

void cb_afficher_decrypt_vigenere(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Récupération du texte.
— Appel de la fonction ni_espace_ni_accent sur le texte.
— Appel de la fonction decryptageSubstitution sur le texte sans espace obtenu précédemment. À cette étape, il y a une circulation l'information entre l'interface graphique et le decryptage. L'interface envoie le texte décrypté et la clé potentielle puis décryptage retourne le texte décrypte.
— Affichage du texte décrypté et la clé potentielle, demande de sauvegarde : si oui échange entre interface et gestion de fichier.

void cb_analyse_frequence(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Ajout d'un widget de saisie.
— Ajout du bouton analysé.
— Connexion du bouton a la fonction cb_affiche_frequence.

void cb_afficher_frequence(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Récupération du texte saisie.
— Ajout de deux widgets pour les cordes.
— Appel aux fonctions frequence_simple et frequence_double. Échange d'informations entre l'interface graphique et le décryptage, interface envoie le texte grâce à l'argument de la fonction et reçoit la fréquence grâce à la valeur retourner par la fonction.
— Affichage des fréquences, demande de sauvegarde : si oui un échange d'informations entre l'interface et la gestion des fichiers est effectué.

void cb_coffre_fort(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Ajout des boutons nouveau coffre et mes coffres.
— Correction des boutons aux fonctions cb_nouveau_coffre et cb_mes_coffres.

void cb_nouveau_coffre(GtkWidget *pwidget, gpointer user_data) :
— Création de la fenêtre.
— Ajout des champs nom et mot de passe.
— Ajout du bouton validé.
— Valider sera connecté à une fonction qui déplacera le fichier dans le dossier créé. Échange d'informations entre interface et gestion de fichier.
void cb_mes_coffres(GtkWidget *pwidget, gpointer user_data) :
— Création de fenêtre.
— Affichage de tous les dossiers, sélection et modification de fichier possible (suppression, chiffrement, déchiffrement et décryptage).

5 Explication du fonctionnement de l'application ODD

L'application possède plusieurs caractéristiques :
- Chiffrement (par Substitution et Vigenère). - Déchiffrement (par Substitution et Vigenère). - Décryptage (par Substitution et Vigenère). - Analyse des Fréquences. - Coffre fort (sauvegarde de données).

L'utilisateur possède plusieurs options pour son utilisation, outre les différents outils à sa disposition pour chiffrer/déchiffrer/décrypter/faire l'analyse des fréquences de ses textes, il pourra également sauvegarder ses activités grâce à la fonctionnalité "Coffre-fort" accessible via l'interface.

Amélioration possible de l'application
En plus de toutes les fonctionnalités présentées plus haut, l'application pourrait être potentiellement améliorée en offrant la possibilité à l'utilisateur d'avoir un compte personnel. Grâce à ses identifiants créés en amont, l'utilisateur pourrait avoir accès à l'application et à son coffre-fort dès son identification faite. L'utilisateur pourrait sauvegarder à son envie son travail dans des fichiers qui auront été créés dans l'interface graphique après qu'il ait sélectionné un bouton qui sauvegardera ce dernier dans le coffre-fort.

Il pourra aussi utiliser l'application sans identifiant (en appuyant sur le bouton "Utiliser sans compte") c'est-à-dire qu'il aura accès aux mêmes fonctions que les inscrits hormis le coffre-fort. Pour chaque session lancée sans identifiant, aucun fichier n'est créé après la fermeture de l'application, il sera alors impossible à l'utilisateur de récupérer les données de son travail.

Si l'utilisateur souhaite quand même enregistrer son travail après avoir cliqué sur "Utiliser sans compte", il lui suffira d'appuyer sur le bouton "Enregistrer" après avoir utilisé l'un des modules. Une fenêtre s'affiche pour lui proposer de créer un compte donc de choisir ses identifiants et son mot de passe, afin d'enregistrer ses fichiers dans son coffre-fort.

6 Conclusion

Pour conclure, ce cahier des spécifications nous permet de visualiser les différentes fonctions qui seront utilisées pour notre application (ODD), ainsi que leur fonctionnement.

Nous avons décrit toutes les fonctions et structures de données que nous allons utiliser dans notre application, le lien entre l'organigramme du cahier des charges et les spécifications, les changements effectués sur l'organigramme ainsi qu'une petite explication sur notre réorientation vers le langage C.

Ce cahier permet donc de décrire de manière spécifique les différentes solutions apportées dans le cahier des charges, notamment les données en entrée ou en sortie des fonctions procédures qui composent nos modules.

Il contient les explications détaillées des bibliothèques et de chaque module que nous avons implémenté, le module du chiffrement, déchiffrement, décryptage ainsi que le module de l'interface graphique.