

Des Cryptes

BRUN Dylan - CARON Sebastian - MAGIEU Pierre

24/09/24

Contexte

Dans un monde où la sécurisation des informations est cruciale, les techniques de cryptographie demeurent un pilier essentiel pour protéger les données personnelles et professionnelles. Bien que des méthodes modernes de chiffrement comme AES ou RSA soient largement utilisées aujourd'hui, les techniques de cryptographie classiques comme le code Morse, le chiffrement de César et le chiffre de Vigenère jouent encore un rôle important dans l'enseignement des bases de la cryptographie et de la sécurité informatique. Elles permettent de comprendre les concepts fondamentaux du chiffrement, comme le décalage des caractères, la substitution et l'utilisation de clés.

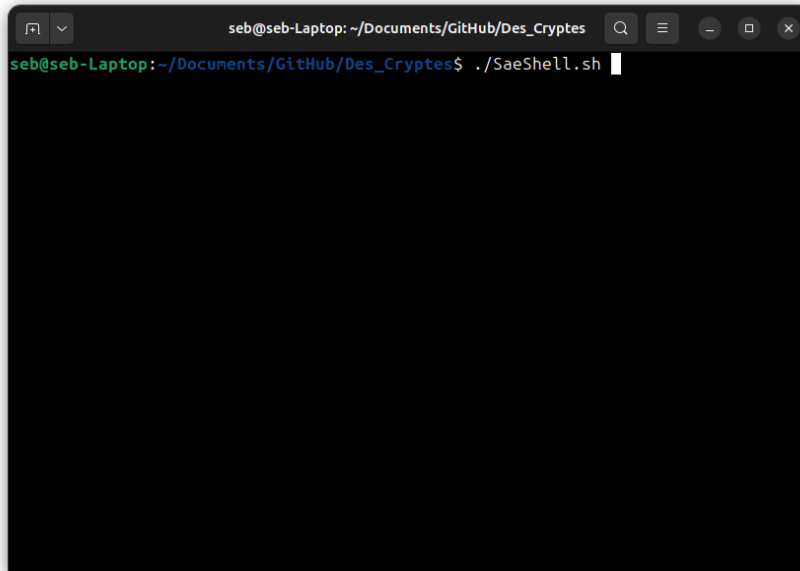
Le projet *Des Cryptes* offre un outil accessible et interactif pour l'application de ces méthodes classiques. En plus d'être un support pédagogique, ce projet offre une expérience pratique avec un script en ligne de commande qui permet aux utilisateurs de chiffrer et déchiffrer des textes, soit en saisissant des chaînes de caractères, soit en travaillant sur des fichiers texte complets.

Table des Matières

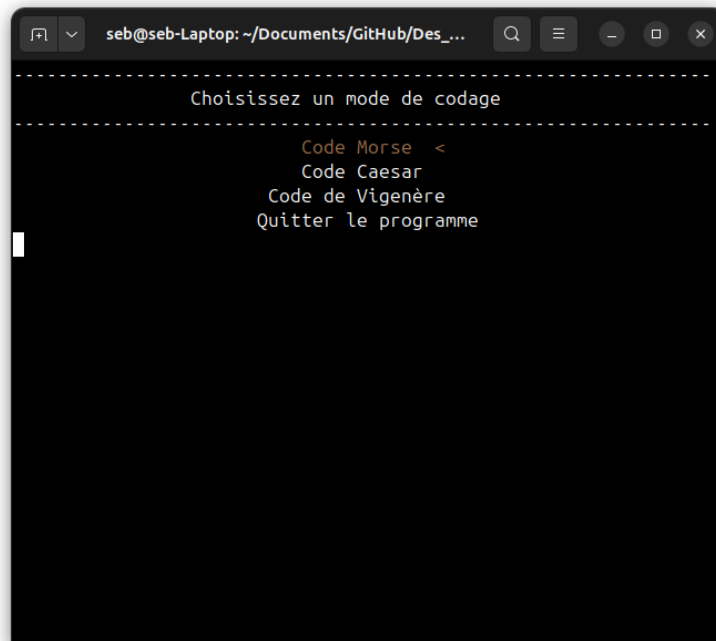
Des Cryptes	1
Contexte	2
Table des Matières	3
Lancement du Projet	4
Caesar	5
Description	5
Programme	6
Exemples	8
Morse	9
Simple Digression Algorithmique	13
Chargement du fichier JSON : morse.json	14
Exemples	15
Vigenere	16
Description	16
Programme	17
Exemples	20

Lancement du Projet

Il suffit de lancer le script SaeShell.sh sans paramètre. Ce script est le point d'entrée de notre projet, il affiche un menu de sélection entre les différents moyens de chiffrements disponibles (Caesar, Morse et Vigenère) et demande à l'utilisateur son choix. Une fois le choix fait, l'utilisateur sera redirigé vers le moyen de chiffrement correspondant.



```
seb@seb-Laptop: ~/Documents/GitHub/Des_Cryptes
seb@seb-Laptop:~/Documents/GitHub/Des_Cryptes$ ./SaeShell.sh
```



```
seb@seb-Laptop: ~/Documents/GitHub/Des_...
-----
Choisissez un mode de codage
-----
Code Morse <
Code Caesar
Code de Vigenère
Quitter le programme
```

Caesar¹

Description

Le chiffrement de César est une méthode cryptographique qui utilise une clé, représentée par un nombre, pour chiffrer et déchiffrer un texte. Cette technique repose sur un décalage appliqué à chaque caractère en fonction de la clé. Elle fonctionne aussi bien pour les lettres minuscules, majuscules et les chiffres.

Chiffrement

Pour chiffrer une lettre, on récupère son code ASCII auquel on ajoute la valeur de la clé. Par exemple, si l'on souhaite chiffrer la lettre 'a' avec une clé de chiffrement égale à 3, le code ASCII de 'a' étant 97. En ajoutant la valeur de la clé à 97, on obtient 100, ce qui correspond à la lettre 'd' dans la table ASCII.

Si le résultat de cette addition dépasse le code ASCII de 'z' (lorsqu'on traite des lettres minuscules), on utilise l'opérateur modulo pour rester dans la plage des lettres minuscules. Cela permet de revenir au début de l'alphabet et d'éviter de dépasser les limites des caractères alphabétiques.

Prenons maintenant l'exemple de la lettre 'a' avec une clé de chiffrement égale à 146. Après avoir ajouté la clé, le résultat dépasse largement le code ASCII de 'z'. En appliquant un modulo 26 (le nombre de lettres dans l'alphabet) sur cette valeur, on ramène le résultat dans la plage des lettres minuscules. En ajoutant cette valeur ajustée à celle de 'a', on obtient alors la lettre 'j'.

Le même principe s'applique pour les lettres majuscules et les chiffres, avec des ajustements en fonction de la plage voulue dans la table ASCII. Les autres caractères, comme les symboles et espaces, ne sont pas modifiés et sont laissés intacts.

Déchiffrement

Le déchiffrement avec la méthode de César fonctionne de la même manière que le chiffrement, à la différence près qu'au lieu d'ajouter la clé au code ASCII de chaque caractère, on la

¹ https://fr.wikipedia.org/wiki/Chiffrement_par_d%C3%A9calage

soustrait. Cela permet de retrouver le texte original en inversant le décalage appliqué lors du chiffrement. Les mêmes règles s'appliquent pour les majuscules, minuscules, chiffres, ainsi que pour la gestion des dépassements avec l'opérateur modulo. Les caractères non-alphanumériques restent inchangés.

Programme

La fonction **CaesarMain** appelle la fonction **affichageCaesar**, qui gère l'affichage du menu principal. Cette dernière est appelée à chaque fois qu'un menu est affiché.

Ce menu principale propose 4 choix :

- Coder
- Décoder
- Retour aux choix de codage
- Quitter

Coder appelle la fonction **CaesarChif**, qui affiche les choix suivants :

- Coder une phrase : Coder directement une phrase saisie par l'utilisateur en utilisant la fonction **codeDecodeCaesar**.
- Coder le contenu d'un fichier : appelle la fonction **creerFichierCaesar**, qui crée un fichier pour stocker le texte codé. Le fichier de l'utilisateur peut être traité en entier ou ligne par ligne avec la fonction **codeDecodeFichierCaesar**.
- Choisir une clé de codage : Permet de choisir une clé soit aléatoirement, soit en la saisissant manuellement (la clé doit être un nombre entier).
- Retour : permet de retourner au menu principal du codage Caesar en appelant **CaesarMain**.

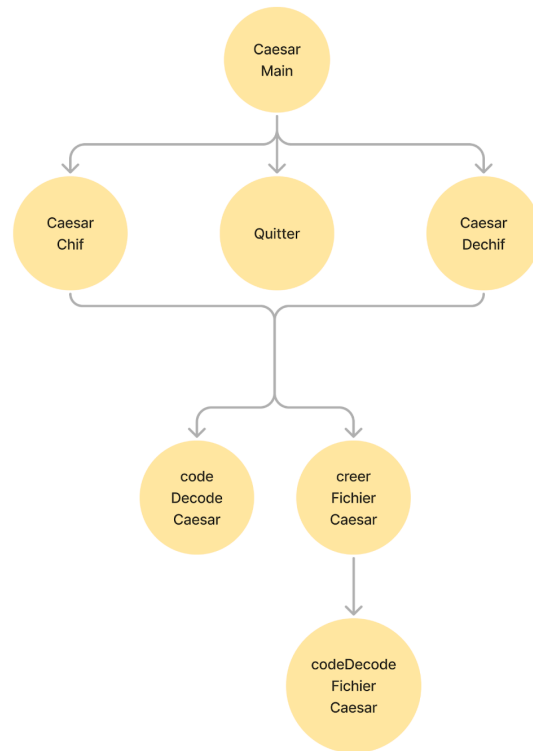
Décoder appelle la fonction **CaesarDéchif** qui propose un menu avec les choix suivants :

- Décoder une phrase : Décoder une phrase saisie par l'utilisateur en utilisant la fonction **codeDecodeCaesar**.
- Décoder le contenu d'un fichier : Décoder tout ou partie d'un fichier avec **creerFichierCaesar**, de la même manière que pour le codage.
- Retour : Retourne au menu principal.

Table des fonctions :

Nom	Input	Output
<i>caesarMain</i>		Affichage du Menu Principal pour Caesar.
<i>caesarChif</i>		Affichage du Menu Principal pour Caesar pour chiffrer.
<i>caesarDechif</i>		Affichage du Menu Principal pour Caesar pour déchiffrer.
<i>codeDecodeCaesar</i>	Chaine de caractère à coder/décoder, clé de décodage (si besoin)	Code ou décode une phrase entrée en paramètre dans la sortie standard.
<i>codeDecodeFichierCaesar</i>	Chemin du fichier, clé de décodage (si besoin), le numéro de la première ligne à coder/décoder (si besoin), numéro de la dernière ligne à coder/décoder (si besoin)	Code ou décode le contenu du fichier entré en paramètre dans un fichier de sortie.
<i>creerFichierCaesar</i>	Chemin du fichier, variable pour savoir si le fichier doit être codé ou décodé	Crée le fichier d'output où il y aura le texte codé/décodé.
<i>affichageCaesar</i>		Gère l'affichage des éléments du menu.
<i>menu</i>	"3" car le menu à 3 choix, rien si le menu à 4 choix	Gère l'interaction entre le menu et l'utilisateur.

Figure 1 : Diagramme explicatif du fonctionnement de la partie Caesar :



Exemples

```
peio@peio-laptop: ~/Documents/GitHub/Des_Cry...  
-----  
Que voulez vous faire ?  
-----  
Coder <  
Décoder  
Choisir un autre mode de codage  
Quitter le programme
```

```
peio@peio-laptop: ~/Documents/GitHub/Des_Cry...  
-----  
Vous avez choisi de coder une phrase  
Entrez la phrase que vous souhaitez coder...  
Hello world  
-----  
Voici votre phrase codée  
Ifmmp xpsme  
-----  
Que souhaitez-vous faire ?  
-----  
Coder le contenu d'un fichier externe <  
Coder une phrase  
Changer la clé de codage, elle est égale à 1  
Retour au menu Caesar
```

```
peio@peio-laptop: ~/Documents/GitHub/Des_Cry...  
-----  
Vous avez choisi de décoder une phrase  
Entrez la phrase que vous souhaitez décoder..  
Hello World  
-----  
Entrez la clé pour décoder cette phrase  
1  
-----  
Voici votre phrase décodée  
Gdkkn Vnqkc  
-----  
Que souhaitez-vous décoder ?  
-----  
Le contenu d'un fichier externe <
```


Morse²

Le code Morse est un code permettant de transmettre un texte à l'aide de signaux courts ou longs.

La fonction **MorseMain** appelle l'affichage qui propose différents choix à l'utilisateur: le chiffrement, déchiffrement et retourner au menu principal. Comme le montre le diagramme de la figure 1., il s'agit du point d'entrée pour toutes les fonctionnalités de chiffrement et déchiffrement du morse.

Selon le choix, retourne au menu ou appelle la fonction **MorseMenu** avec un paramètre : 0 pour le chiffrement, 1 pour le déchiffrement.

La fonction **MorseMenu** appelle l'affichage qui propose différents choix à l'utilisateur : chiffrer ou déchiffrer en utilisant un fichier externe, via l'entrée standard ou de retourner au menu précédent.

Si l'utilisateur choisit via l'entrée standard alors la fonction appelle **MorseInput**.

Si l'utilisateur choisit via un fichier externe alors la fonction appelle **MorseFile**, puis **DechiffrementFichierMorse** ou **ChiffrementFichierMorse** selon l'option.

La fonction **MorseInput** chiffre ou déchiffre selon le paramètre avec lequel elle a été appelée : 0 pour le chffrement, 1 pour le déchiffrement.

La fonction lit les lignes via l'entrée standard et les chiffre ou les déchiffre à moins qu'une ligne soit égale à "/exit" auquel cas, elle retourne au menu principal.

La fonction **MorseFile** va demander à l'utilisateur un fichier d'entrée valide et un fichier de sortie valide. Les chemins vers les fichiers vont être stockés dans des variables.

Les fonctions **DechiffrementFichierMorse** et **ChiffrementFichierMorse** sont quasi identiques, elles vont toutes les deux lire le fichier d'entrée ligne par ligne, appliquer la fonction de chiffrement ou de déchiffrement sur chacune des lignes, puis écrire le résultat dans le fichier de sortie.³

La fonction **codeMorse** va récupérer la chaîne de caractères avec laquelle elle a été appelée en paramètre. La fonction va, pour chaque caractère de la chaîne, tester si le caractère est dans le dictionnaire initialisé au lancement du fichier [morse.sh](#) avec les valeurs du fichier

² https://fr.wikipedia.org/wiki/Code_Morse_international

³ NB : **dechiffrementFichierMorse** et **chiffrementFichierMorse** ne sont absolument pas responsable de l'existence ou de la validité des fichiers d'entrée et sortie. Ce travail est délégué à la fonction **MorseFile**.

morse.json. Si le caractère est dans le dictionnaire alors il est codé, sinon il est laissé tel quel, puis concaténé à une chaîne de caractères résultats qui est affichée une fois tous les caractères parcourus.

La fonction **decodeMorse** fonctionne exactement de la même manière excepté qu'elle lit les éléments blocs par blocs où les blocs sont séparés par des espaces.

Lorsqu'il y a une erreur ou quelque chose à notifier à l'utilisateur, par exemple lorsque le fichier d'entrée ou de sortie est invalide, la variable globale erreur est mise à jour. Cette valeur varie entre 1 et 3 :

- 1 = Choix invalide dans la sélection au menu
- 2 = Le fichier choisi n'existe pas
- 3 = Le fichier a été sauvegardé avec succès
- Autre Valeur = Aucune erreur

L'affichage des menus entre **MorseMain** et **MorseMenu** est très similaire. Une fonction a donc été écrite afin de factoriser notre code : **AffichageScreen**.

AffichageScreen affiche donc un titre et des options tant que l'utilisateur n'a pas effectué un choix valide. En addition à cet affichage s'ajoute un appel à la fonction **ErreurFunc** qui affiche ou non un message à l'utilisateur selon la valeur de la variable erreur.

Afin de respecter le cahier des charges, une fonction récursive a été ajoutée :

DecodeMorseRecursive. Cette fonction récursive est appelée par **DecodeMorseRecursiveStart** qui initialise la récursion.

DecodeMorseRecursiveStart prend en paramètre une chaîne de caractères, la découpe pour la transformer en liste puis appelle **DecodeMorseRecursive** avec l'indice courant à 0, la taille de la liste, le résultat actuel ainsi que les éléments de la liste.

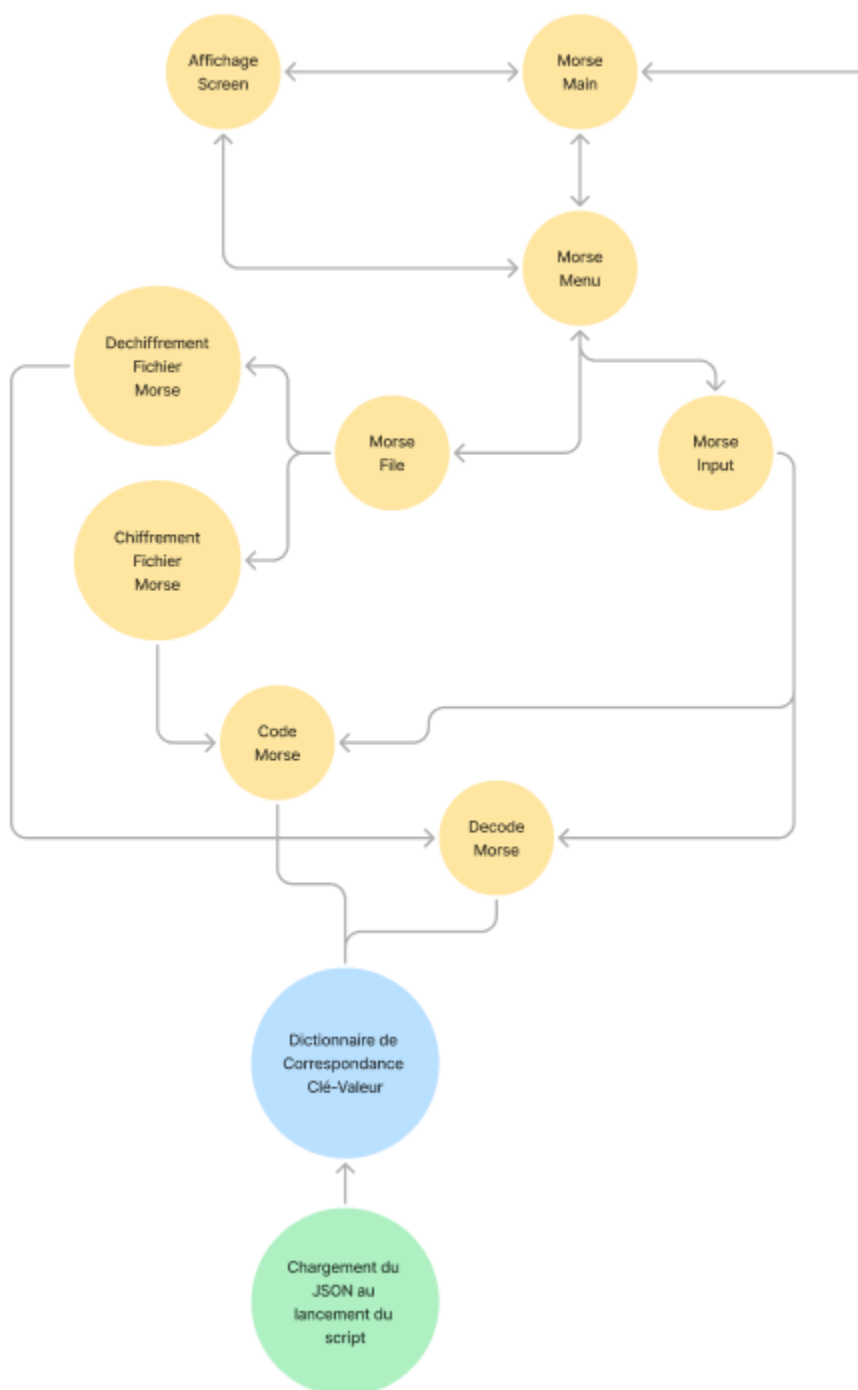
DecodeMorseRecursive va ainsi récupérer l'élément à l'indice courant, décoder l'élément, l'ajouter au résultat et passer à la récursion suivante avec l'indice incrémenter. La récursion continue jusqu'à ce que l'indice soit égale à la taille de la liste auquel cas, l'entièreté de la liste a été parcourue.

DecodeMorseRecursiveStart et **DecodeMorseRecursive** sont écrites mais jamais utilisées.

Table des fonctions :

Nom	Input	Output
MorseMain		Affichage du Menu Principal pour Morse.
MorseMenu	0 pour le chiffrement ou 1 pour le déchiffrement.	Affichage du Menu pour la sélection du mode : via un fichier ou via l'input console.
MorseInput	0 pour le chiffrement ou 1 pour le déchiffrement.	Récupère les lignes dans la console et chiffre ou déchiffre selon le paramètre.
MorseFile	0 pour le chiffrement ou 1 pour le déchiffrement.	Demande un fichier d'entrée valide et un fichier de sortie valide. Appelle ensuite DechiffrementFichierMorse ou ChiffrementFichierMorse .
DechiffrementFichierMorse	Un chemin vers un fichier d'entrée et un chemin vers un fichier de sortie.	Aucune sortie mais écrit dans le fichier de sortie les lignes du fichier d'entrées converties.
ChiffrementFichierMorse	Un chemin vers un fichier d'entrée et un chemin vers un fichier de sortie.	Aucune sortie mais écrit dans le fichier de sortie les lignes du fichier d'entrées converties.
codeMorse	Une chaîne de caractères.	La chaîne convertie en morse.
decodeMorse	Une chaîne de caractères.	La chaîne convertie en langage naturel.
ErreurFunc		L'erreur selon le code d'erreur global.
AffichageScreen		Affiche le titre, les options et met à jour le choix selon l'option sélectionnée.
DecodeMorseRecursiveStart	Une chaîne de caractères.	La chaîne convertie en langage naturel.
DecodeMorseRecursive	L'indice courant, la taille de la liste, le résultat, la liste.	Les éléments de la liste convertis en langage naturel concaténés dans le résultat.

Figure 2 : Diagramme explicatif du fonctionnement de la partie Morse :



Simple Digression Algorithmique

Une manière simple de stocker la correspondance "caractère : morse" est d'utiliser deux listes : une qui stocke les caractères, une qui stocke la valeur morse correspondante.

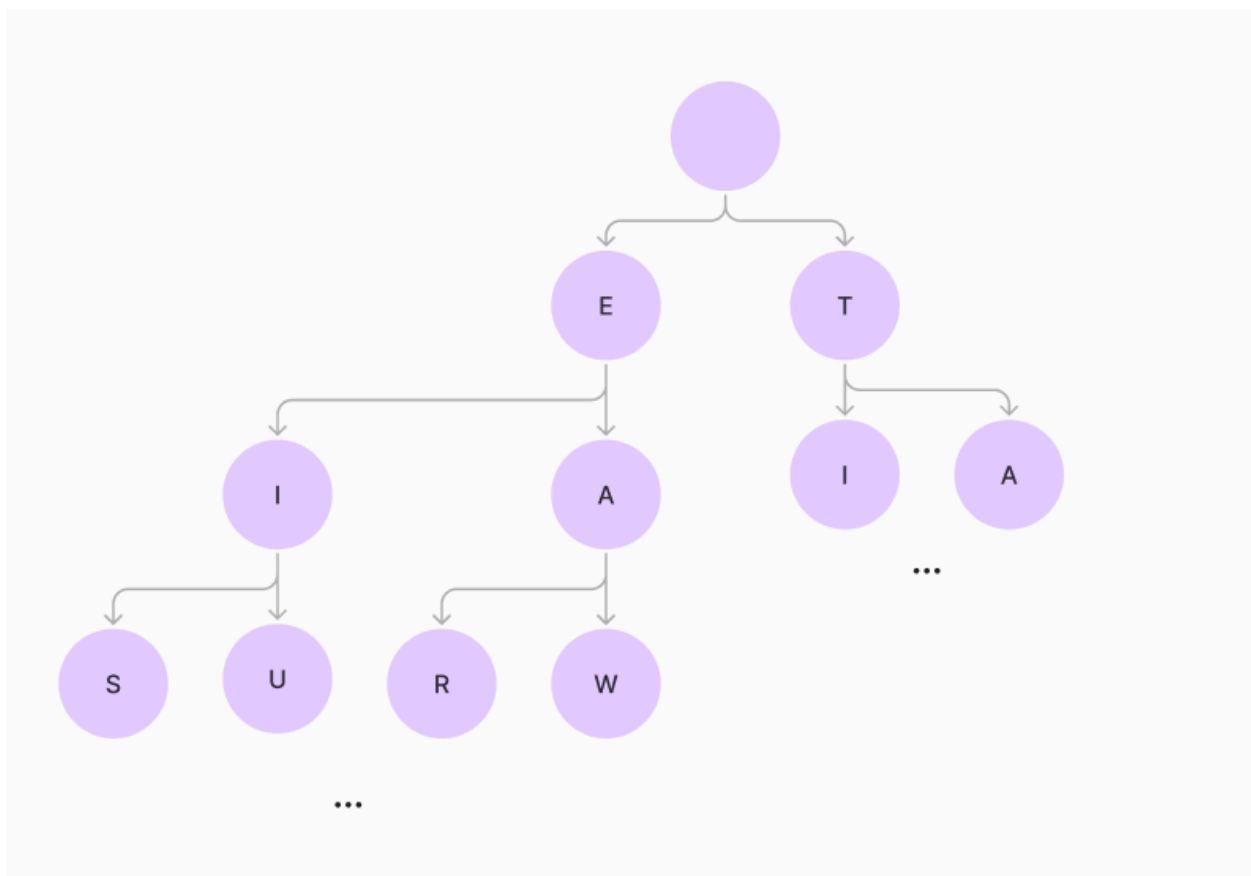
```
chars = ["A", "B", "C", "D", ...]  
morses = ["._", "_...", "_._.", "_..._", ...]
```

Pour coder ou décoder, il suffit simplement de parcourir les éléments et si c'est l'élément que l'on cherche, on récupère l'indice et la valeur correspondante se trouve au même indice dans l'autre liste.

Il s'agit d'une manière très simple de stocker les données mais on code ou décode la valeur en $O(n)$, car dans le pire des cas la valeur n'est pas dans la liste. Pas très efficace mais suffisant...

Une autre manière de représenter le Code Morse est d'utiliser un Arbre Binaire :

Figure 2 : Arbre Binaire représentant le code Morse.



Si il y a un court ".", on va vers l'enfant gauche. Si il y a un long "_", on va vers l'enfant droit.

Ainsi ".._" donnera : gauche, gauche, droite. Ce qui code le caractère "U".

NB : le caractère " " (espace) est codé par "/" et n'est donc pas présent dans l'arbre.

En utilisant un arbre binaire, le décodage est donc possible en $\log_2(n)$ mais le codage se fait toujours en $O(n)$. En effet, pour coder un caractère, il suffit d'explorer l'arbre (DFS, BFS ...). Cependant si le caractère ne possède aucun Code Morse correspondant alors on parcourt l'entièreté de l'arbre.

Une autre approche utilise des tables de hachages. On applique une fonction de hachage à notre valeur ce qui nous donne son adresse.

Une mauvaise fonction de hachage nous donnera accès à notre valeur correspondante en temps linéaire tandis qu'une excellente fonction de hachage (peu/pas de collisions) nous donnera accès à notre valeur correspondante en temps quasi-constant / constant.

Ainsi le codage et décodage en Morse se fait en $O(1)$.

Bash4 supporte nativement les listes associatives⁴, c'est donc cette option qui a été utilisée.

S'il n'y avait pas eu d'implémentation native de table de hachage, notre implémentation aurait résolu les collisions par chaînage.

Chargement du fichier JSON : morse.json

Le fichier morse.json est lu ligne par ligne.

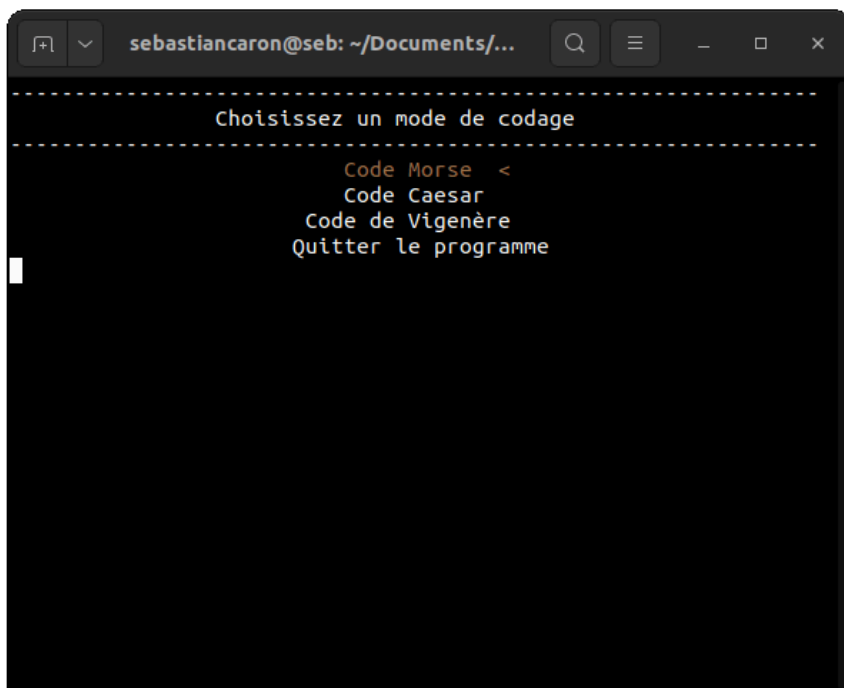
Si la ligne match cette REGEX : `"([^\"]+)"[[:space:]]*:[[:space:]]*"([^\"]+)"`

alors le premier et second groupe sont récupérés à l'aide de `BASH_REMATCH`⁵ et ajoutés dans les listes associatives.

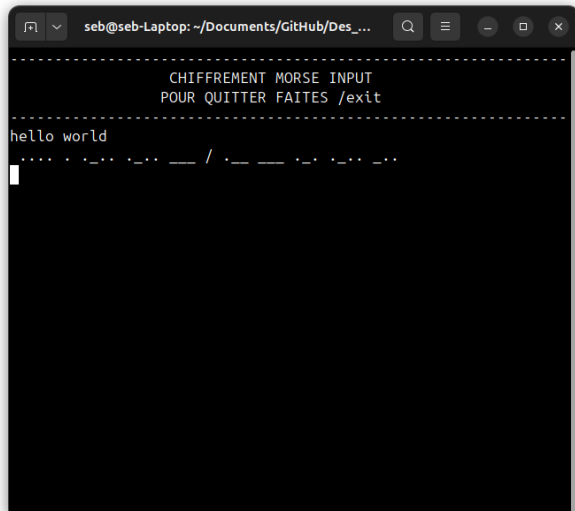
⁴ https://www.gnu.org/software/bash/manual/html_node/Arrays.html

⁵ https://www.gnu.org/software/bash/manual/html_node/Bash-Variables.html

Exemples

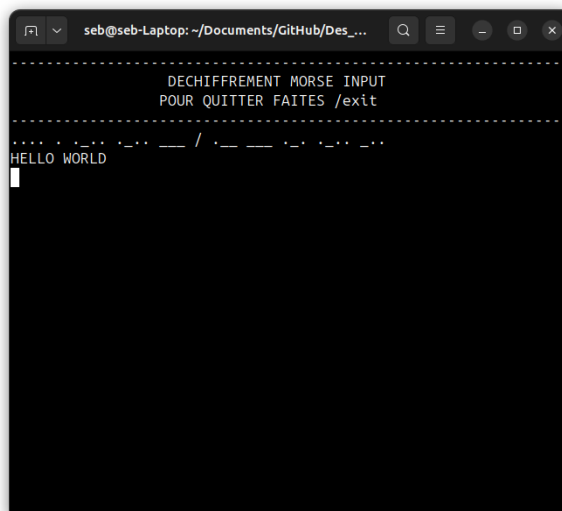


```
sebastiancaron@seb: ~/Documents/...
-----
Choisissez un mode de codage
-----
Code Morse <
Code Caesar
Code de Vigenère
Quitter le programme
```



```
seb@seb-Laptop: ~/Documents/GitHub/Des_...
-----
CHIFFREMENT MORSE INPUT
POUR QUITTER FAITES /exit
-----
hello world
.... . _... _... _ / _ _ _ _ _ _ _ _

```



```
seb@seb-Laptop: ~/Documents/GitHub/Des_...
-----
DECHIFFREMENT MORSE INPUT
POUR QUITTER FAITES /exit
-----
.... . _... _... _ / _ _ _ _ _ _ _ _
HELLO WORLD

```

Vigenere

Description

Le chiffrement de Vigenère est une méthode de cryptographie qui utilise une clé pour chiffrer et déchiffrer un texte. Contrairement au chiffrement de César, le chiffrement de Vigenère utilise une clé composée de plusieurs lettres pour appliquer le décalage sur chaque caractère du texte.

Le fonctionnement repose sur une répétition de la clé jusqu'à la fin du texte. À chaque lettre du texte, on associe une lettre de la clé, et le décalage appliqué à la lettre du texte dépend de la de l'index de la lettre de la clé dans l'alphabet (0-25). Si une lettre de la clé est en majuscule ou en minuscule, elle est traitée comme une minuscule.

Dans cette édition du chiffrement de Vigenère, chaque chiffre des nombres sera chiffré.

Chiffrement

Pour chiffrer une lettre, on la déplace dans l'alphabet selon l'index de la lettre de la clé. Par exemple, si la lettre de la clé est "B" (la deuxième lettre de l'alphabet), la lettre du texte sera déplacée de 1 position vers la droite. Si la clé est plus courte que le texte à chiffrer, elle est répétée autant de fois que nécessaire.

Déchiffrement

Le déchiffrement fonctionne de la même manière mais inverse l'opération en décalant chaque lettre du texte vers la gauche selon la lettre correspondante de la clé. Chaque lettre de la clé détermine combien de positions la lettre chiffrée doit être déplacée dans l'alphabet pour retrouver la lettre originale.

Exceptions

Tous caractères n'étant pas alphanumériques ne seront pas pris en compte lors du chiffrement et du déchiffrement d'un texte et resteront à leur état d'origine.

Programme

Partie Principale

En premier lieu, la fonction `vigenereMain` permet d'initialiser toutes les variables globales, puis appelle la fonction `vigenereMain_` qui s'occupe de l'affichage du premier menu (le menu principal).

Dans le menu principal, il y a 4 choix:

- Chiffrer
- Déchiffrer
- Retour
- Quitter

Chiffrer

Chiffrer appelle la fonction `chiffrerVigenere` qui propose un menu avec les choix suivants:

- Choisir une clé
- Utiliser une clé générée (aléatoirement)
- Retour
- Quitter

Retour permet de retourner dans le menu principal (en appelant `vigenereMain`).

Quitter utilise une fonction du programme `tools.sh` qui arrête complètement le code.

Choisir une clé appelle la fonction `choixCle` qui permet de proposer à l'utilisateur une clé de son choix (les caractères hors alphabétiques ne seront pas utilisés lors du chiffrement/déchiffrement).

Utiliser une clé générée comme son nom l'indique appellera la fonction `genCle` qui générera une clé de façon (de taille 1 à 69 choisit aléatoirement).

Dans le cas où vous avez choisi une clé (générée aléatoirement ou pas),

Déchiffrer

Déchiffrer appelle la fonction `dechiffrerVigenere` qui propose le menu suivant:

- Choisir une clé
- Retour
- Quitter

Comme pour le chiffrement, Retour appelle la fonction `vigenereMain` pour aller au menu principal, et Quitter appelle la fonction `quitter` de `tools.sh` qui arrête le programme, et Choisir une clé appelle la fonction `choixCle` qui va vous permettre d'insérer la clé de votre choix.

Résultat

Suite au choix de la clé, les fonctions `estFileInput` et `estFileOutput` vont être appelées pour demander à l'utilisateur s'il veut utiliser un fichier en tant qu'entrée et en tant que sortie.

Si un fichier est entré, alors on va demander le chemin du fichier via la fonction

`choixFichierInput`, puis si on veut chiffrer/déchiffrer une ligne ou le fichier en entier avec

`selectLigne`. Pour le fichier en sortie, on demande son chemin et si l'utilisateur veut écraser ou ajouter la chaîne de caractères à la fin du fichier avec `choixFichierOutput`.

Une fois fait, on chiffre/déchiffre avec `chiffrementVigenere` et `dechiffrementVigenere`, et on demande si l'utilisateur veut quitter le programme avec la fonction `continuerYN`, et s'il souhaite continuer on appelle la fonction `vigenereMain`.

(La plupart des erreurs sont traités par la fonction `actionInvalide`)

Table des fonctions:

Nom	Input	Output
vigenereMain		Initialise toutes les variables
vigenereMain_		Affiche le menu principale
actionInvalide		Affiche un message d'erreur
continuerYN		demande à l'utilisateur s'il veut continuer
estFileInput	un chemin (chaîne de caractères)	Vérifie si le chemin est un fichier, sinon en créer un
estFileOutput	un chemin (chaîne de caractères)	Vérifie si le chemin est celui d'un fichier, sinon créer en un
choixFichierInput		demande le nom du fichier en entrée
choixFichierOutput		demande le nom du fichier en sortie
selectLigne		Demande quelle ligne doit être chiffrée/déchiffrée
choixCle		Attribue à la variable cle le choix de l'utilisateur
choixPhrase		Demande une phrase à l'utilisateur
genCle		Génère une clé aléatoirement
chiffrementVigenere	une chaîne à chiffrer, et une clé	Chiffre et affiche la chaîne de caractères
dechiffrementVigenere	une chaîne à déchiffrer, et une clé	Déchiffre et affiche la chaîne de caractères
chiffrerVigenere		Menu du chiffrement
dechiffrerVigenere		Menu du déchiffrement

Exemples

```
seb@seb-Laptop: ~/Documents/GitHub/Des_...  
-----  
Veuillez choisir une action  
-----  
Chiffrer <  
Dechiffrer  
Retour  
Quitter
```

```
seb@seb-Laptop: ~/Documents/GitHub/Des_...  
-----  
Veuillez choisir une action  
-----  
Choisir une clé <  
Utiliser une clé générée  
Retour  
Quitter
```

```
seb@seb-Laptop: ~/Documents/GitHub/Des_...  
Voici la clé: vthhetlpufmgkxwanga  
Voici la phrase chiffrée: ntsbx  
-----  
Voulez-vous quitter le programme ?  
-----  
Oui <  
Non
```

Diagramme du programme *vigenere.sh*