# Projet en C

## Parsing de mots

Sagara

# Table des matières

# Chapitre 1

# Introduction

Le but principal du projet est d'acquérir et de développer des connaissances en C. Il sera demander d'avoir un code propre, clair et commenté, ainsi que d'avoir un dépôt Git distant bien entretenu ( c'est-à-dire que le code sur le dépôt doit être fonctionnel ). L'objectif du projet est de créer un parseur de mots. Tout le long du projet, vous serez guidés par de nombreuses sources fournies. Si une question, ou un problème se pose veuillez la poser à l'adresse suivante : sagarafr@gmail.com

## 1.1 Généralités

### 1.1.1 Parseur

Un parseur est un outil d'analyse syntaxique. Il permet de mettre en évidence une structure particulière de fichier et de pouvoir repérer facilement des données.

### 1.1.2 Git

Git est un outil de gestion de versions décentralisé, Il permet facilement de récupérer et de faire évoluer un projet collaboratif accessible depuis l'Internet.

### 1.1.3 Recommandation

Vu que nous utilisons un langage bas niveau (proche du langage machine), les différents programmes devront se terminé correctement et en libérant toute la mémoire allouée par celui-ci. Aucune fuite mémoire et autres erreur de segmentation sont accepté et acceptable.

# Chapitre 2

# Une étape primordiale

La première étape afin de pouvoir facilement créer un parseur est d'avoir un structure de donnée de type String. Une String est une chaîne de caractère et que vu que nous allons analyser des mots, il est primordial d'avoir une structure de données pouvant gérer ce type facilement. En C, ce type n'existe pas nativement. Même si des bibliothèques nous permettent de le gérer, nous allons vous demander de le programmer, afin que vous appreniez à mieux gérer les tableaux et les structures en C.

## 2.1 Les chaînes de caractères

En C, les Strings ne sont pas implémentées de manière native. Il est donc important de pouvoir les programmer et de les utiliser correctement. Pour cela, on utilise un tableau de caractères afin de pouvoir stocker chaque lettres d'un mot dans une case du tableau. Les chaînes de caractères ont pour norme de terminer par ce caractère spécial : \0.

## 2.2 La structure utilisée

La structure a utilisée pour gérer les chaînes de caractères est la suivante :

```
1 struct string
2 {
3     unsigned int length;
4     char* string;
5 };
6 typedef struct string String;
```

Elle nous permet de stocker la longueur de la chaîne de caractères ainsi que sont contenu. Comme dit plus haut, chaque mot a un caractère terminal, il ne faut pas le compter dans la longueur de la chaîne mais il faut lui allouer de la mémoire afin qu'il prenne la dernière place du tableau de caractères.

## 2.3 Les fonctions utilisant cette structure

Pour faciliter l'utilisation de cette structure, nous allons implémenter différentes fonctions.

```
void create_string(String* s);
```

Cette fonction permet d'initialiser la champs de la structure String aux valeurs suivantes :
- length doit être à 0.
- string doit être à NULL.

```
void free_string(String* s);
```

Cette fonction permet de libérer la mémoire prise, avec la commande free, par le tableau de chaîne de caractère, et de réinitialiser les champs de la structure par les valeurs par défaut.

```
void add_character(String* s,const char c);
```

Cette fonction permet de rajouter le caractère c, à la fin de la chaîne de caractère contenu dans la variable s. Il faut donc allouer ou réallouer de la mémoire afin d'agrandir le tableau de chaîne de caractère, via respectivement malloc et realloc, rajouter le caractère manuellement et mettre à jour la longueur de la chaîne de caractère. On peut connaître la taille mémoire d'un type grâce à sizeof.

```
void add_string(String* s,const char* c);
```

Cette fonction permet de rajouter la chaîne de caractère c à la fin de la chaîne de caractère contenu dans la variable s. Il faut donc allouer ou réallouer correctement la mémoire afin d'agrandir le tableau de chaîne de caractère, via malloc et realloc, rajouter la chaîne de caractère via strcpy et mettre à jour la longueur de la chaîne de caractère. On peut savoir la longueur de la chaîne c avec la fonction strlen et connaître la taille mémoire d'un type grâce à sizeof.

```
void print_string(const String* const s);
```

Cette fonction permet d'afficher la chaîne de caractère contenu dans s, avec printf.

```
void create_p_string(String** s);
```

Cette fonction permet d'allouer de la mémoire, si nécessaire, au pointeur de type String et d'initialiser les champs de la structure de donnée avec les valeurs par défaut. On peut connaître la taille mémoire d'un type grâce à sizeof.

```
void free_p_string(String** s);
```

Cette fonction permet de libérer totalement la mémoire prise par le pointeur de la structure de donnée. Cela signifie qu'il faut libérer la mémoire prise par le pointeur de la structure et du tableau pris par la chaîne de caractère. Enfin de d'initialiser les champs de la structure par des valeurs par défaut.

# Chapitre 3

# La gestion de fichier

Nous allons analyser, ou parser, des fichiers. Il nous faut donc un moyen pour les ouvrir et les lire facilement. Les différents bibliothèques standards C le permettent et nous allons vous expliquer comment les utiliser, afin de pouvoir lire et écrire. Nous donnerons des fichiers de tailles respectables afin qu'ils tiennent entièrement en mémoire. Dans cette partie, il vous sera demandé de pouvoir fournir un bout de programme permettant d'ouvrir, de lire et de stocker les données dans le type String et de fermer le fichier.

# Chapitre 4

# Utilisation des arbres

Maintenant que l'on sait récupérer des données depuis un fichier, nous allons maintenant créer une structure de données particulière pour les stocker : les arbres. Cette structure offre la possibilité de pouvoir facilement rechercher et de stocker des mots. Ainsi dans cette étape, il sera demander de créer des arbres équilibrés et de les stocker dans un fichier avec un formalise spécifique.

# Chapitre 5

# Un peu de compression en plus

Vu que maintenant, nous savons analyser un fichier et stocker les éléments importants dans un fichier, nous allons maintenant utiliser les différents outils produits afin de pouvoir stocker nos informations en prenant un minimum de place. Nous verrons dans cette partie le codage de Huffman, qui nous permettra de compresser nos différentes données.