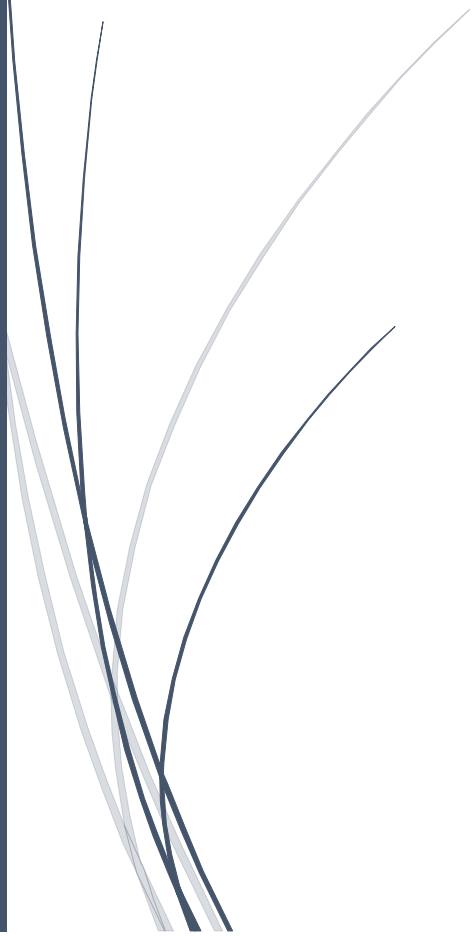
20/11/2022

Exquisitus Cadaveris

« Une ovalbumine ? Détestablement. »



Gabriel DURIEUX
Paul ZANOLIN
Albane COIFFE

Table des matières

Introduction

Le projet de ce semestre consistait en un petit programme nommé cadavre exquis. Le principe ? générer des phrases en utilisant des mots aléatoires. Pour cela, nous avons dû utiliser des mots d'un dictionnaire, et les accorder en suivant le contexte de la phrase afin d'obtenir des phrases cohérentes mais n'ayant aucun sens.

L'objectif principal était de nous familiariser avec l'utilisation des arbres en C en les utilisant dans un projet concret. On a pu notamment les utiliser à des fins de recherche, pour trouver des mots de manière efficaces, et en utilisant des arbres N-aires plutôt que des arbres binaires.

Présentation fonctionnelle

Génération de phrases

Le but principal de ce projet est de générer plusieurs types de phrases. Pour cela, on sélectionne un mot au hasard dans le dictionnaire et on accorde sa forme en fonction. Deux modèles différents de phrase ont été proposés dans l'énoncé du projet, et nous devions en créer un autre nous-même. Nous avons donc créé des fonctions qui étaient capables de chercher l'accord d'un mot avec des conditions données, comme le temps, le genre ou le nombre. Grâce à cela, il nous est possible de générer un mot aléatoire respectant certaines conditions, et de former ensuite une phrase. Nous avons aussi réalisé une fonction qui permet d'associer un article à un nom.

Recherche d'un mot

Les mots du dictionnaire étaient stockés sous forme d'arbre. Il fallait donc être capable de trouver un mot dans cet arbre. Nous avons donc réalisé cette fonctionnalité. Elle se présente sous deux formes différentes. La première, « Rechercher un mot », est ici plus utilisée à des fins de démonstration. Elle permet de trouver l'adresse de la structure qui stocke un certain mot dans l'arbre. La seconde, utilise la même base mais se sert cette fois de l'adresse du mot en question pour afficher ses différentes formes fléchies.

Fonctionnalités supplémentaires

Nous avons réalisé deux fonctionnalités supplémentaires, qui ont principalement été utilisées lors du développement. La première permet d'afficher un arbre, ce qui nous a été très utile pendant le développement pour vérifier si tout fonctionnait correctement. La seconde permet d'afficher la graine utilisée pour initialiser l'aléatoire, ce qui nous permet ensuite de la fixer manuellement et de reproduire une situation pour laquelle nous aurions un bug, par exemple.

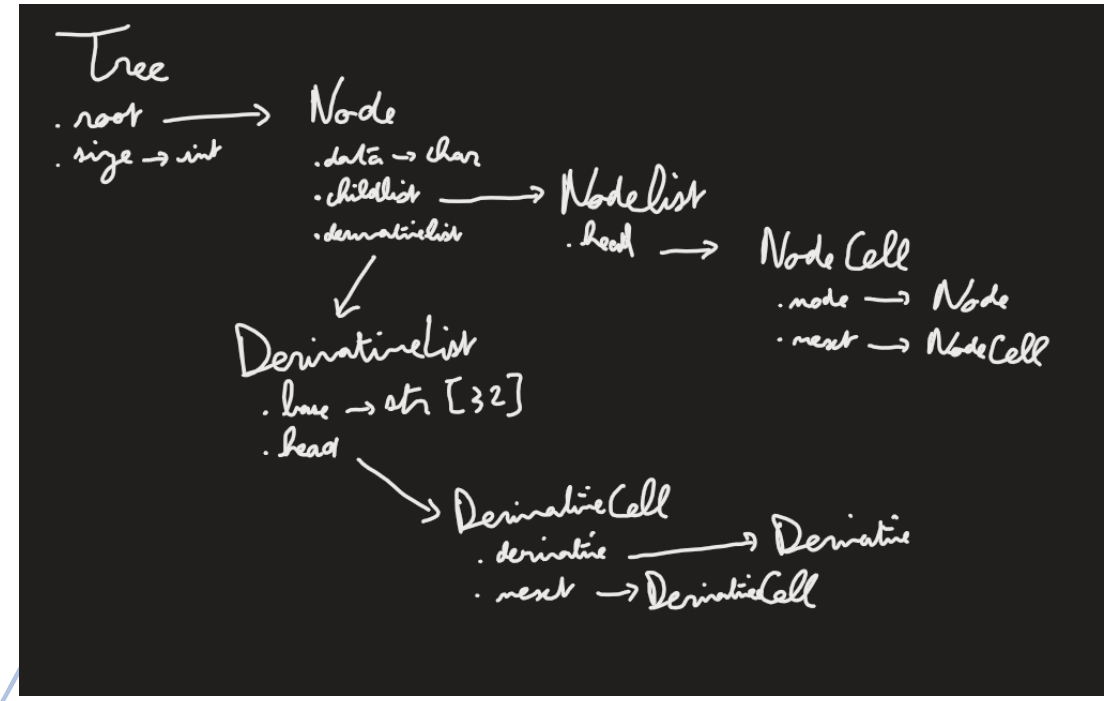
Présentation technique

Structures de données

Nous avons élaboré une structure de données assez complexe pour stocker toutes ces informations dans notre programme. Pour commencer, une structure simple Dictionary nous permet de stocker les différents arbres (adverbes, noms, verbes et adjectifs) (structure ci-contre).

```c
typedef struct dictionary {
    struct Tree * trees[4];
} Dictionary;
```

Les arbres sont constitués d'un nœud racine. Ensuite, chaque nœud possède une valeur (en l'occurrence une lettre) ainsi qu'une liste chainée de nœud enfant, ce qui nous permet de faire des arbres N-aires et non binaires. Les nœuds qui symbolisnet la fin d'une forme de base voit leurs champs DerivativeList rempli des formes flechies ainsi que leurs descritptions, ce qui permet, si ce champ pointe vers NULL, de déduire que ce nœud n'est pas celui de fin d'une forme de base. La structure est décrite plus en détail dans le shémas suivant.



Concernant les dérivatives, cet objet nous permet de décrire une forme fléchie tel que représenté ci-dessous.

```c
typedef struct Derivative {
    enum type { ADV, NOM, VER, ADJ, DETR, PRON, PREP, CONJ, ABRV, INTJ, ONOM, QPRO, dtype} type;
    enum gender { MAS, FEM, INVGEN, CARD, dgender } gender;
    enum tense { IPRE, IFUT, IPSIM, IIMP, SIMP, SPRE, INF, CPRE, PPRE, PPAS, IMP, IMPRE, dtense }  tense;
    enum person { P1, P2, P3, dperson } person;
    enum number { SG, PL, INVPL, dnumber } number;
    char word[32];
} Derivative;
```

Algorithmes

Nous avons utilisé les algorithmes vus en cours pour interagir avec notre structure de données. Évidemment, il a fallu les remanier légèrement pour d'adapter à notre structure, mais dans l'ensemble ils restent similaires à ceux que l'on a étudié. Par exemple, les fonctions de création d'objet suivent toutes à peu près le même schéma : une allocation mémoire, si besoin certaines valeurs sont mises à leurs valeurs par défaut, puis on retourne le pointeur vers l'objet créé. (Exemple ci-contre). Une des fonctions qui a été la plus complexe a développer était processDerivative. Cette fonction a pour but de prendre en parametre la forme flechie d'un mot ainsi que sa description, et de donner a l'objet Derivative les propriétés nécessaires. Popur cela, nous séparons les configurations entre les :, grace a la foncition strtok, puis entre les + pour découper les propriétés. Cest chaines de caractères sont ensuite traitées par des sous fonctions, qui déterminent si les valeurs doivent etre celles par défaut ou des valeurs identifiées. Petit bonus, nous avons utilisé la propriété des « cases », qui est que sans break, toutes les instructions suivantes sont éxecutées. Cela permet d'ajouter deux propriétés a verifier pour les verbes seulement par exemple. Le code de la fonction est le suivant :

```c
Tree * createTree() {
    Tree * tree = malloc( Size: sizeof(Tree));
    tree->root = createNode( data: ' ');
    tree->size = 0;
    return tree;
}
```

```c
Derivative * processDerivative(char * word, char * form) {
    Derivative * derivative = createEmptyDerivative();
    char * currentDerivation, * currentParameter;
    derivative->type = getType(form);
    if(derivative->type >= NOM && derivative->type <= PRON) {
        while((currentDerivation = strtok( Str: NULL, Delim: ":"))) {
            currentParameter = strtok( Str: currentDerivation, Delim: "+");
            do {
                switch (derivative->type) {
                    case VER:
                        derivative->tense = derivative->tense == dtense ? getTense( tense: currentParameter) : derivative->tense;
                        derivative->person = derivative->person == dperson ? getPerson( person: currentParameter) : derivative->person;
                    case NOM:
                    case ADJ:
                    case DETR:
                        derivative->gender = derivative->gender == dgender ? getGender( gender: currentParameter) : derivative->gender;
                        derivative->number = derivative->number == dnumber ? getNumber( number: currentParameter) : derivative->number;
                        break;
                    default:
                        // ici atterit tout ce qui est sans paramètres avancés
                        // ADV, PREP, INTJ, CONJ, CONN, ABRV, ONOM, QPRO
                        break;
                }
            } while ((currentParameter = strtok( Str: NULL, Delim: "+")));
        }
    }
    strncpy( Dest: derivative->word, Source: word, Count: 32);
    return derivative;
}
```

Difficultés

Malgré un développement plutôt sans embuche, nous avons quand même fait face a quelques difficultés. Premièrement, l'élaboration de la structure de donnée fût un challenge important,

car ce point est la clé de voute du programme. En cas de modification de la structure principale, c'est l'entièreté du programme qui est affecté. Cela nous est arrivé une fois, heureusement vers le début du développement. Nous avons aussi rencontré beaucoup de bugs que nous avons dû régler, souvent à cause d'erreurs d'interprétation de notre part concernant la structure complexe du programme. Nous avons aussi eu un problème majeur avec le GitHub, et des changement incohérents entre eux ont été commit, ce qui nous a valu une bonne heure de débogage pour permettre à notre programme de compiler à nouveau. Enfin, Nous avons dû développer un petit algorithme pour analyser le dictionnaire, afin de découvrir les différents types et valeurs possibles. Voici le script ainsi que son résultat.

```
with open("dictionnaire_non_accentue.txt", 'r') as f:
    lines = f.readlines()
result = {}
for line in lines:
    mode = line.split("\t")[2]
    splited = mode.split(":")
    formType = splited[0].strip()
    if len(formType) <= 4 or "+" not in mode:
        firstPartLen = len(formType)
        if formType not in result:
            result[formType] = []
        if len(splited) > 1:
            for s in splited[1:]:
                plusSplited = s.split("+")
                for t in plusSplited:
                    t = t.strip()
                    if t not in result[formType]:
                        result[formType].append(t)
print(result)
```

Résultat :

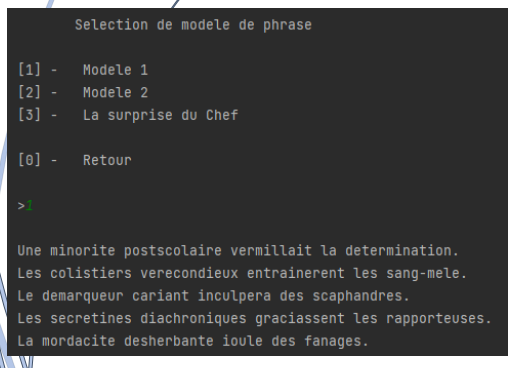
{'Nom': ['Mas', 'InvPL', 'SG', 'Fem', 'PL', 'InvGen', 'Card', 'Nom'], 'Pre': [], 'Ver': ['IPre', 'SG', 'P3', 'IPSim', 'P1', 'IImp', 'PL', 'P2', 'PPre', 'SImp', 'SPre', 'ImPre', 'PPas', 'Mas', 'Fem', 'IFut', 'Inf', 'CPre', 'Imp', 'P3(Ãˆ C0QÃˆ)', ''], 'Adj': ['InvGen', 'SG', 'Mas', 'Fem', 'PL', 'InvPL', 'Card'], 'Adv': [], 'Int': [], 'Det': ['Mas', 'SG', 'Fem', 'PL', 'InvGen'], 'Pro': ['Mas', 'SG', 'Fem', 'PL', 'P3', 'P3PL', '3Fem', '3Mas', 'P1', 'InvGen', 'P2'], 'Con': [], 'Abr': [], 'Ono': [], 'QPro': [], 'Conj': []}

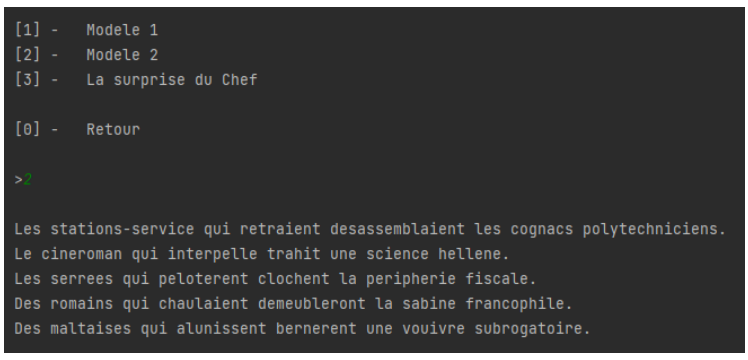
Présentation des résultats

Lorsque le programme se lance, une bannière ainsi qu'une petite citation générée par le programme s'affiche.

Dans le menu principal, on peut choisir de générer des phrases ou rechercher un mot. (L'autocomplétions ne fonctionne pas pour le moment, et les fonctionnalités maudites sont simplement des petites fonctionnalités supplémentaires)

Lorsque l'on sélectionne la première option, on a accès a un menu permettant de choisir le modèle de phrase à générer.





```
       Selection de modele de phrase

[1] -   Modele 1
[2] -   Modele 2
[3] -   La surprise du Chef

[0] -   Retour

>3

Albane matit Gabriel pendant que l' infraction massa inexactement des tire-bouchons sans paddy.
Albane concedera Gabriel pendant que des clochers surevalueront exhaustivement les dessaisissements sans praseodyme.
Albane farina Gabriel pendant que les suppliciees rifleront commodement un verbalisateur sans contrepoids.
Albane desaffecta Gabriel pendant que une tachycardie denoua objectivement des lavasses sans ophtalmologue.
Albane papillotat Gabriel pendant que les tessons collaient doux un neologisme sans forain.
```

La recherche de mot permet elle de choisir entre deux possibilités : Trouver un mot ou afficher ses formes fléchies.

```
       Rechercher dans le dictionnaire

[1] -   Rechercher un mot
[2] -   Rechercher un mot et ses formes flechies

[0] -   Retour

>1
       Entrez le mot a rechercher :
>poule

       "poule" found @ 0x27dff9a8
       tree: Noms
```

```
       Rechercher dans le dictionnaire

[1] -   Rechercher un mot
[2] -   Rechercher un mot et ses formes flechies

[0] -   Retour

>2
       Entrez le mot a rechercher :
>autruche

       autruche -->    autruche autruches
       tree: Noms
```

Enfin, les fonctionnalités maudites permettent d'afficher la graine de l'aléatoire ou les arbres entiers.

```
       ATTENTION ZONE DANGEREUSE

[1] -   Afficher arbre /!\
[2] -   Afficher la seed /!\

[0] -   Retour

>2

       seed: 1668983331
```

```
       ATTENTION ZONE DANGEREUSE

[1] -   Afficher arbre /!\
[2] -   Afficher la seed /!\

[0] -   Retour

>1
      [a[b[jectement, o[minablement, ndamment], r[eviativement, uptement],
quement, c[e[lerando, ssoirement], identellement], r[ement, imonieusement,
, equatement, iabatiquement, jectivement, mi[nistrativement, ra[blement, ti
], ff[ablement, ectueusement, i[nement, rmativement], reusement], g[i[lemen
, lleurs, mablement, nsi, sement], l[e[ntour, rtement], gebriquement, ias,
nt], ors, phabetiquement, t[ernativement, ierement]], m[bi[gument, tieuseme
 plement], n[a[l[ogiquement, ytiquement], rchiquement, tomiquement], cienne
```

Conclusion

Les projets à réaliser en groupe sont toujours des sources d'amélioration des connaissances techniques et humaines.

En effet, d'un point de vue technique, nous avons pu manier les arbres dans un cas concret, ce qui est toujours plus motivant et stimulant. L'utilisation de GitHub nous permet de continuer de nous familiariser avec cet environnement. L'apprentissage via des projets est plus dynamique. Le sujet était aussi très drôle et concret, on voyait directement les résultats de notre algorithme et cela nous motivait à avancer sur le développement. Enfin, en créant d'autres modèles de phrase, nous pouvons y apporter une touche personnelle.

Ce projet nous a permis d'apprendre à travailler à trois, nous avons dû nous organiser. Dans notre groupe, nous avons particulièrement privilégié le pair-coding (ou plutôt trio-coding dans notre cas). De plus, nous devions réaliser ce projet en trois semaines, il y a donc fallu prendre en compte rapidement la contrainte de temps et estimer correctement le temps de développement nécessaire.

Ainsi, ce projet nous a permis de continuer de développer de nombreux soft-skills requis dans le monde du travail tel que la communication au sein du groupe, l'organisation et la répartition des taches en fonction des qualités de chacun, puis améliorer notre aisance à l'oral à travers la soutenance et notre capacité de synthèse lors de la rédaction du rapport. Nous conclurons sur la phrase suivante, qui résume à notre sens bien notre projet : « Un marécage siliceux insufflera les stewards. »