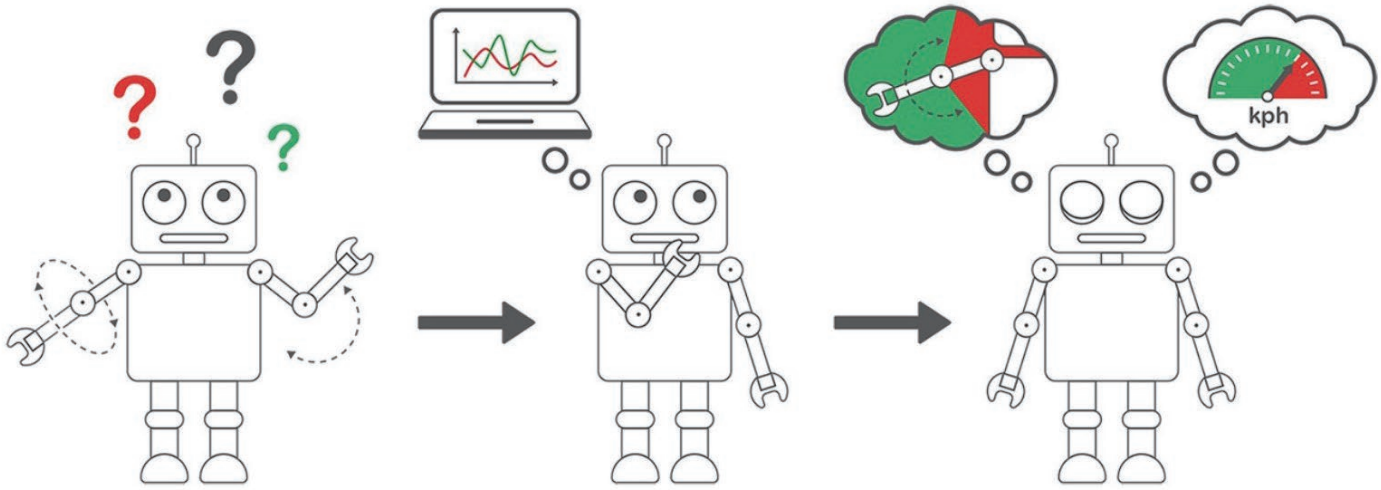
eFrei
PARIS PANTHÉON-ASSAS UNIVERSITÉ



Projet programmation en C :

Générateur de phrases

Pei LIU
Arthur TOUZIN
Issa Kane
Saison 2022-2023 semestre 3

Sommaire

Introduction 1

Partie A : Les structures de données et l'arbre

– Importation du fichier 4

- Création de l’arbre n-aire 5

- Le stockage des mots 12

Partie B : Construction des phrases grammaticalement correctes

- Sélection des mots 13

- Construction de la phrase avec des mots de base 15

- Application des règles grammaticales à la phrase 17

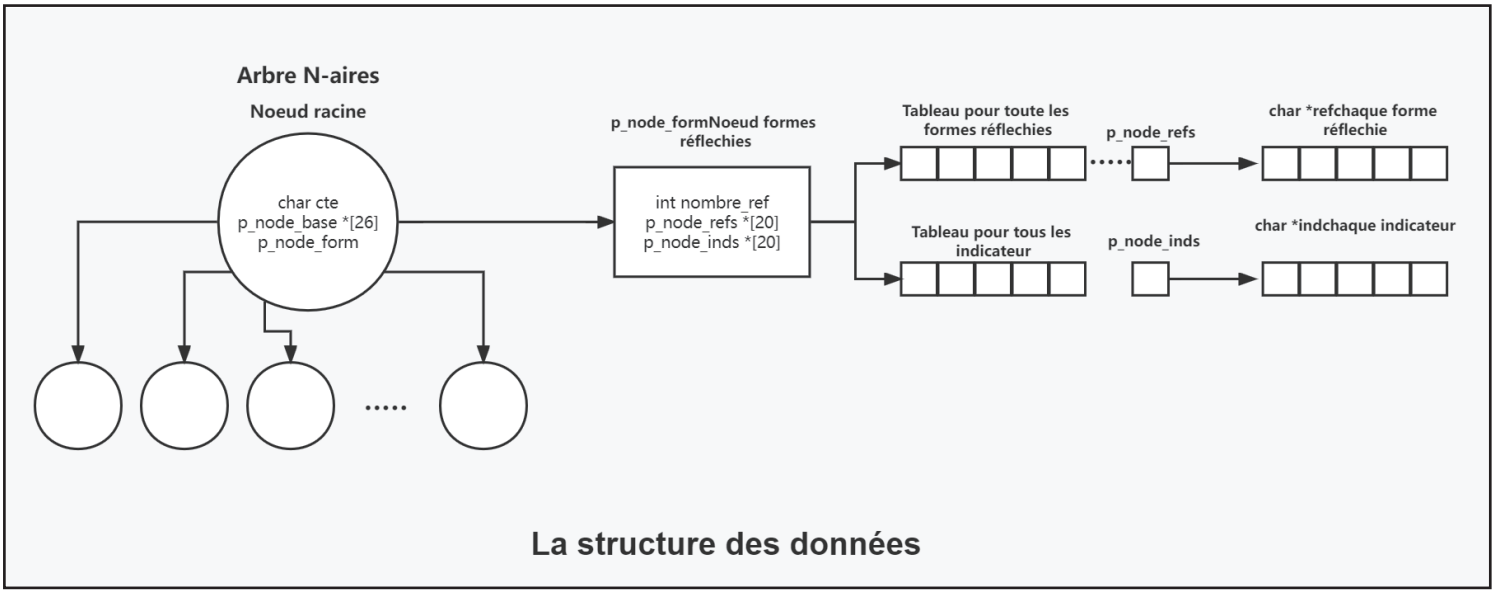
Conclusion 35

Introduction

La programmation trouve des applications pratiques dans tous les domaines notamment dans la littérature. En effet, le surréalisme est défini comme étant un : « jeu qui consiste à faire composer une phrase, ou un dessin, par plusieurs personnes sans qu'aucune d'elles puisse tenir compte de la collaboration ou des collaborations précédentes.» Donc, elle se révèle comme un outil indispensable pour solutionner des problèmes de cet ordre. L'utilisation de l'outil informatique permet d'obtenir des résultats précis en un temps minimum.

La réalisation de ce projet comporte deux axes majeurs. Il s'agit d'une part de programmer des codes relatifs à la structure de données et l'arbre n-aires. Et d'autre part, nous devront mettre en place des systèmes pour solutionner des problèmes d'ordre littéraire en l'occurrence de la création de phrases grammaticalement correctes.

Partie A: Les structures des données et l'arbre



La structure des données

Pour stocker les mots, on a crée des arbres n-aires pour chaque catégorie de mots.

Chaque nœud d'arbre contient 3 éléments : la lettre correspondante, 26 sous-arbres contenant les 26 lettres de l'alphabet et la structure qui va stocker les formes fléchies.

La structure des formes fléchies contient :

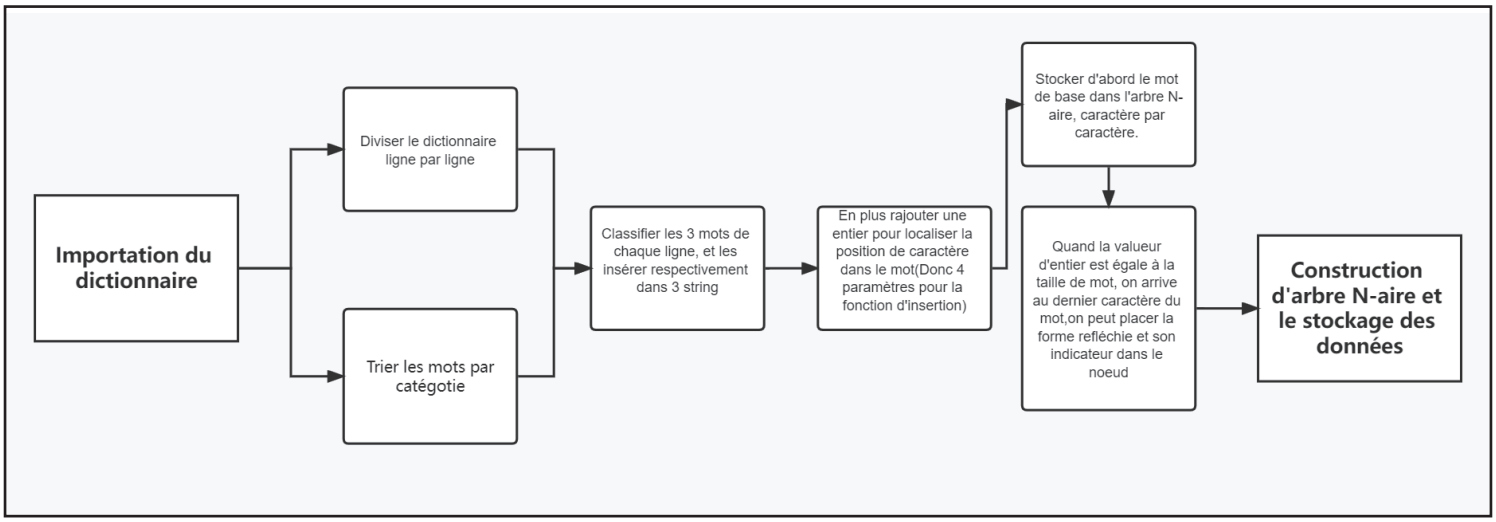
Un entier pour le nombre de formes du mot

Un tableau pour stocker toutes les formes fléchies

Un tableau pour stocker les indicateurs

(La position sur le tableau de l'indicateur correspond à la position sur le tableau de formes fléchies.)

- Importation du fichier



Pour importer le fichier.txt, on a créé un buffer circulaire pour prendre le contenu du fichier utilisé les fonctions de la bibliothèque stdlib.h :

- fopen pour ouvrir le fichier dans le programme

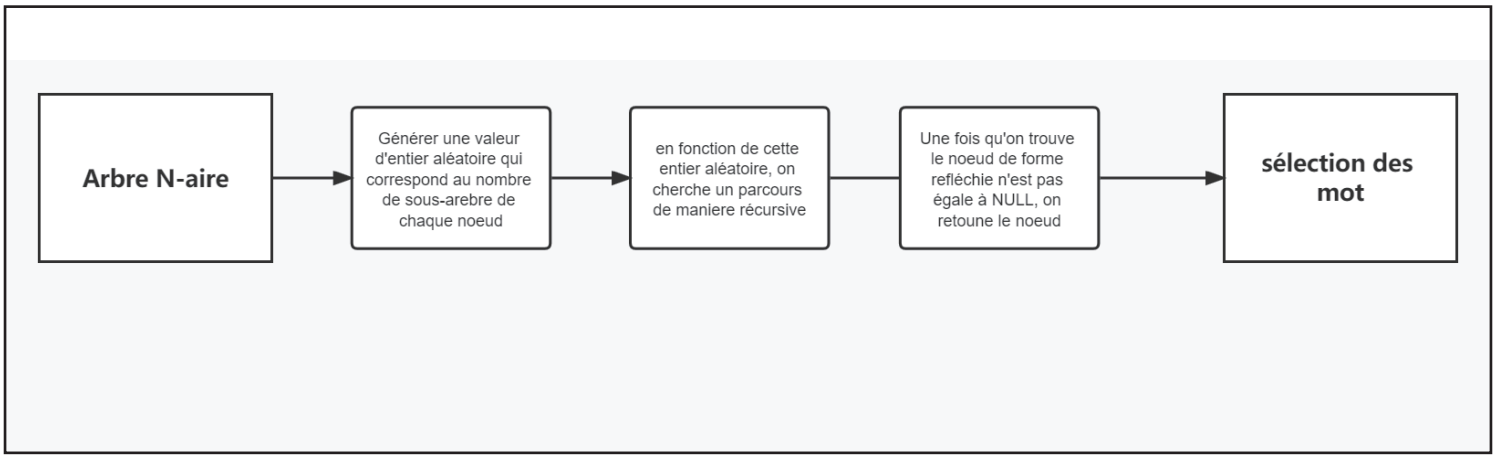
- fgets pour importer le contenu du dictionnaire ligne par ligne dans le buffer

- strtok(buf , “\t“) pour diviser les mots en 3 parties (Formes fléchis, forme de base et indicateur de formes fléchis).

Ci-dessous vous verrez le code bien commenté

```
//Buffer
char buf[MAX_LINE];
FILE *fp; // file pointer


const char *ref;  //stocker la forme réflechie
const char *base; //stocker le mot de base
const char *ind;  //stocker l'indicateur
t_n_tree tree = createTree( val: '\0');


//Ouverture du fichier
if((fp = fopen( Filename: "C:\\Users\\31799\\Desktop\\dic.txt", Mode: "r")) == NULL){

    perror( ErrMsg: "Fail to read");
    exit( Code: 1);
}


while (!feof( File: fp)){
    int count = 3;

    memset( Dst: buf, Val: 0,  Size: sizeof (buf));
    fgets( Buf: buf, MaxCount: sizeof(buf) - 1,  File: fp); //importer le dictionnaire dans le buffer ligne par ligne
    char *temp = strtok( Str: buf,  Delim: "\t");
```

- Le stockage des mots et la création des arbres



On va stocker les mots dans l'arbre par catégorie en fonction de l'indicateur de mot, on aura un arbre pour les verbes avec l'indicateur (Ver), un arbre pour les noms avec l'indicateur (Nom), un abre pour les adjectifs avec l'indicateur (Adj) et un arbre pour les adverbes (Adv).

```
// créer l'arbre n-aires par catégorie
if(ind[0] == target) {

    if(ind[2] == 'j'){
        for (int i = 0; i < MAX; i++) {
            if (((tree.root)->son)[i] == NULL) {
                ((tree.root)->son)[i] = createBaseNode( val: '\0');
                insertNode(((tree.root)->son)[i], ref, base, ind, 0);
                break;
            } else if (base[0] == ((tree.root->son)[i])->cte) {
                insertNode(((tree.root)->son)[i], ref, base, ind, 0);
                break;
            }
        }
```

Comme écrit dans le code on a catégorisé les mots via la troisième lettre de l'indicateur.

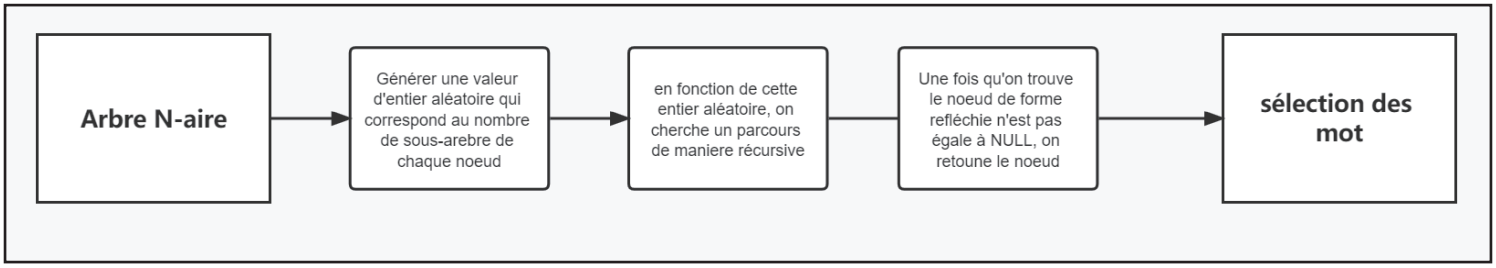
Par exemple Adj : Mas + SG on a pris 'j' pour savoir qu'il s'agit d'un adjectif.

Pour insérer les nœuds on a créé une fonction qui s'appelle insertNode qui prend 4 paramètres soit le mot de base, la forme fléchies, l'indicateur et un entier pour localiser chaque caractère du mot.

Voici le code bien détaillé

```
//insertion des nodes dans l'arbre
void insertNode(p_node_base pnb,char *ref,char *base,char *ind,int position){

    //dans le cas, l'insertion de mot n'est pas arrivée à la fin du mot
    if(position != strlen( Str: base)-1){
        if(pnb->cte == '\0'){  //si le noead est vide
            pnb->cte = base[position];

            for(int i = 0; i < MAX;i++){
                if ((pnb->son)[i] == NULL) { //si le sous-arbre de ce noead est NULL
                    (pnb->son)[i] = createBaseNode( val: '\0');
                    insertNode( pnb: (pnb->son)[i], ref, base, ind,  position: position + 1); //continuer à inserer le prochain caractère
                    break;
                } else if (base[position + 1] == ((pnb->son)[i])->cte) { //si le caractère existe
                    insertNode( pnb: (pnb->son)[i], ref, base, ind,  position: position + 1);//entrer dans le prochain noead
                    break;
                }
            }
        }else if(pnb->cte == base[position]){ //si le caractère existe
            for(int i = 0; i < MAX;i++){
                if ((pnb->son)[i] == NULL) { //voir le prochain noead s'il y a le même caractère
                    (pnb->son)[i] = createBaseNode( val: '\0');
                    insertNode( pnb: (pnb->son)[i], ref, base, ind,  position: position + 1);
                    break;
                } else if (base[position + 1] == ((pnb->son)[i])->cte) {
                    insertNode( pnb: (pnb->son)[i], ref, base, ind,  position: position + 1);
                    break;
                }
            }
        }
    }else if(position == strlen( Str: base)-1){ //La fin du mot
        if(pnb->cte == '\0'){

            pnb->cte = base[position]; // s'il n'y pas de caractère existant pour le dernier caractère du mot
        }

        if(pnb->ref->amount == 0){  // dans le cas, il n'y a pas encore de forme réflechie stockée

            (pnb->ref->inds)[pnb->ref->amount] = createIndNode( ind: "base");
            (pnb->ref->refs)[pnb->ref->amount] = createRefNode( ref: base);
            pnb->ref->amount++;
            (pnb->ref->inds)[pnb->ref->amount] = createIndNode(ind);
            (pnb->ref->refs)[pnb->ref->amount] = createRefNode(ref);
            pnb->ref->amount++;
        }else{

            (pnb->ref->inds)[pnb->ref->amount] = createIndNode(ind);
            (pnb->ref->refs)[pnb->ref->amount] = createRefNode(ref);
            pnb->ref->amount++;
        }

        return;
    }
```

- La sélection des mots




```
//chercher un noead qui contient les formes flechies de manière aléatoire
p_node_base nodeFind (p_node_base base,int seed){

    int s = rand() % 26 +1;  //la facteur aléatoire pour éviter de choisir des noead qui sont toujours avant

    if(base->ref->amount != 0 && s != 0){

        s--;
    }
    if(base->ref->amount != 0 && s == 0){ //si un noead contient les formes réflechies

        return base;
    }
    p_node_base temp = NULL;
    if(base->son[seed] != NULL){ //par une facteur aléatoire on parcour l'arbre

        temp = nodeFind( base: base->son[seed], seed: rand() % 26);
        if(temp != NULL){
            return temp;
        }
    }else if(base->son[seed] == NULL){

        temp = nodeFind(base, seed: rand() % 26);
    }

    return temp;
}
```

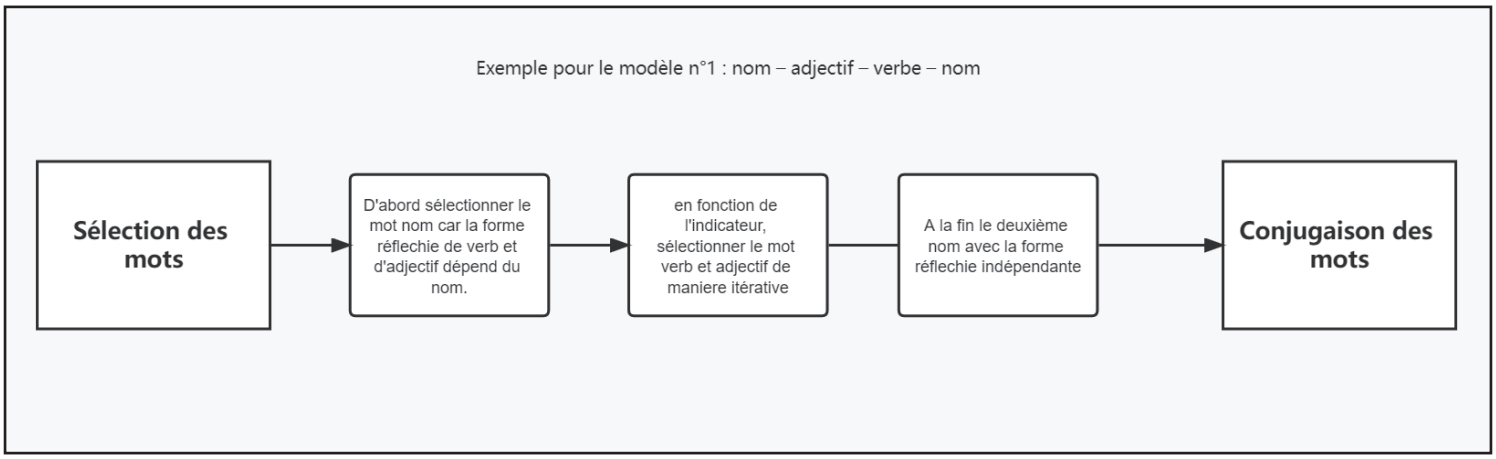
Construire des phrases avec des mots de base :

```
if(pnb->ref->amount == 0){  // dans le cas, il n'y a pas encore de forme réflechie stockée

    (pnb->ref->inds)[pnb->ref->amount] = createIndNode( ind: "base");
    (pnb->ref->refs)[pnb->ref->amount] = createRefNode( ref: base);
    pnb->ref->amount++;
    (pnb->ref->inds)[pnb->ref->amount] = createIndNode(ind);
    (pnb->ref->refs)[pnb->ref->amount] = createRefNode(ref);
    pnb->ref->amount++;
```

Avec la fonction nodeFind (sélection des mots), on peut trouver tous les mots nécessaires de chaque catégorie avec le nœud. Et pendant le stockage des mots on stocke la forme de base dans la première place du tableau (tableau des formes fléchies).

En conséquence, on peut réaliser la construction des phrases avec les mots de bases

- La conjugaison



```
p_n_node initNode(){

    p_n_node nNode = malloc( Size: sizeof (t_n_node));
    nNode->ref = NULL;
    nNode->ind = NULL;

    return nNode;
}
//Trouver une forme réflechie aléatoirement pour un nom
p_n_node toSentence1(p_node_base n_base1){

    p_n_node nNode = initNode();

    int n_seed1 = (rand() % ((n_base1->ref->amount)-1))+1; //La facteur aléatoire qui correspond au nombre de forme réflechie
    nNode->ref = (n_base1->ref->refs)[n_seed1]; //Trouver une forme réfléchie précise
    nNode->ind = (n_base1->ref->inds)[n_seed1];

    if(strstr( Str: nNode->ind->ind, SubStr: "Nom:Mas+SG")){
        if((nNode->ref->ref)[0] == 'a' || (nNode->ref->ref)[0] == 'e' || (nNode->ref->ref)[0] == 'h' || (nNode->ref->ref)[0] == 'o' || (nNode->re
            printf( format: "l'%s ",nNode->ref->ref);
        }else{
            printf( format: "le %s ",nNode->ref->ref);
        }
```

Il faut autogénéré un entier comme un facteur aléatoire qui correspond au nombre de forme fléchis. Et par cet entier on va trouver une forme fléchis précise aléatoirement et retourner la forme fléchie trouvée pour que le verbe et l'adjectif puissent s'en servir pour avoir une forme grammaticale correcte

```
//en fonction de la forme réflechie du nom, chercher une forme pour verb
p_node_base verbFind(p_node_base base,p_n_node nNode,int val1){

    p_node_base n_base;
    while (val1 == 1){ //val1 pour contrôler si on a bien trouvé une forme correspondante dans le noead

        n_base = nodeFind(base,rand() % 26); // trouver un noead dans l'arbre pour verb

        for (int i = (rand() % ((n_base->ref->amount)-1))+1; i < n_base->ref->amount; i++) { //Parcourir toutes les formes

            if (strchr( Str: ((n_base->ref->inds)[i])->ind, Val: (nNode->ind->ind)[strlen( Str: nNode->ind->ind)-2])) {
                if(strstr( Str: ((n_base->ref->inds)[i])->ind, SubStr: "PPas")){
                    if(strstr( Str: nNode->ind->ind, SubStr: "SG")){
                        printf( format: "a %s ",(n_base->ref->refs)[i]->ref);
                    }else if(strstr( Str: nNode->ind->ind, SubStr: "PL")){
                        printf( format: "ont %s ",(n_base->ref->refs)[i]->ref);
                    }
                }else{
                    printf( format: "%s ", (n_base->ref->refs)[i]->ref);
                }
                val1 = 2; //après trouver une forme coresspondante dans le noead, sinon on rentre dans la boucle pour trouver un nouveau noead.
                break;
            }
        }
    }

    return n_base;
}
```

Pour la fonction verbFind, on a mis 3 paramètres, il y a un la racine de l'arbre pour le verbe, la forme fléchie pour le nom qu'on a trouvé et le 3e paramètre pour contrôler si on a trouvé la forme fléchie correspondante du verbe. Pour la suite on va trouver un nœud qui contient des formes fléchies pour le verbe, ensuite on va parcourir aléatoirement les formes fléchies du verbe en fonction de l'entier autogénéré

On fait le même procédé pour les adjectifs

```
//en fonction de la forme réflechie du nom, chercher une forme pour adjectif
p_node_base adjFind(p_node_base base,p_n_node nNode,int val2){

    p_node_base n_base;
    while (val2 == 1){ //val2 pour contrôler si on a bien trouvé une forme correspondante dans le noead
        n_base = nodeFind(base,rand() % 26); // trouver un noead dans l'arbre pour verb

        for (int i = (rand() % ((n_base->ref->amount)-1))+1; i < n_base->ref->amount; i++) { //Parcourir toutes les formes

            if (strrchr( Str: ((n_base->ref->inds)[i])->ind, Ch: (nNode->ind->ind)[strlen( Str: nNode->ind->ind) - 2])) {
                printf( format: "%s ", (n_base->ref->refs)[i]->ref);
                val2 = 2; //après trouver une forme coresspondante dans le noead, sinon on rentre dans la boucle pour trouver un nouveau noead.
                break;
            }
        }
    }
    return n_base;
}
```

- Le fonctionnement du programme

```
//Trouver un nom de manière aléatoire
p_node_base n_base1 = nodeFind(n_tree.root,(rand() % 26));
p_node_base n_base2 = nodeFind(n_tree.root,rand() % 10);

printf( format: "Tapez 1 pour le modèle n°1 : nom - adjectif - verbe - nom \n");
printf( format: "Tapez 2 pour le modèle n°2 : nom - 'qui' - verbe - verbe - nom - adjectif \n");
printf( format: "Tapez 0 pour quitter\n");
printf( format: "Choisissez un modèle de phrase: \n");
scanf( format: "%d",&custm_insert);

if(custm_insert == 0){
    break;
}
else if(custm_insert == 1){

    printf( format: "--> ");
    p_n_node nNode = toSentence1(n_base1); //Trouver la forme réflechie du nom

    p_node_base  adj_base = adjFind(adj_tree.root,nNode,val2); //Trouver un adjectif conjugué de manière aléatoire
    val2 = 1;

    p_node_base verb_base = verbFind(v_tree.root,nNode,val1); //Trouver un verb conjugué de manière aléatoire
    val1 = 1;

    toSentence1(n_base2);

    printf( format: "\n");
    //Imprimer les mots de base
    printf( format: "    (Base: %s %s %s %s) \n\n",(n_base1->ref->refs)[0]->ref,(adj_base->ref->refs)[0]->ref,(verb_base->ref->refs)[0]->ref,(n_base2-
```

En fonction de la valeur entrée par l'utilisateur, on peut avoir différents types de phrases bien structurées

CONCLUSION

En conclusion, ce projet nous a montré que l'informatique était un outil ayant un champ d'action très vaste plus particulièrement le langage C et qu'il peut s'appliquer aussi bien aux mathématiques qu'à la littérature. Aussi, les arbres binaires malgré leurs complexités sont très efficaces en ce qui concerne le stockage des mots. Et le choix de structure de donneé est prend une place importante au nieaux de l'éfficacité et de la faisabilité

PROBLEMES RENCONTREES :

- Malloc probleme de distribution de mémoire (surtout strcpy_s)
- Difficulté à décortiquer le sujet
- Control de complexité et efficacité du programme(entre l'itération et la récursivité)

Annexe

- Le nettoyage du dictionnaire

```
char buf[MAX_LINE];
FILE *fp , *in; // file pointer

//Ouverture du fichier de lecture
if((fp = fopen( Filename: "C:\\Users\\31799\\Desktop\\dictionnaire.txt", Mode: "r")) == NULL){

    perror( ErrMsg: "Fail to read");
    exit( Code: 1);
}

//Ouverture du fichier d'écriture
if((in = fopen( Filename: "C:\\Users\\31799\\Desktop\\dic.txt", Mode: "a")) == NULL){

    perror( ErrMsg: "Fail to read");
    exit( Code: 1);
}

char *temp = malloc( Size: strlen( Str: buf)+1);

while (!feof( File: fp)){

    memset( Dst: buf, Val: 0,  Size: sizeof (buf));
    fgets( Buf: buf, MaxCount: sizeof(buf) - 1,  File: fp);
    strcpy_s( Dst: temp, SizeInBytes: strlen( Str: buf)+1, Src: buf);
    temp = strtok( Str: temp, Delim: "\t"); //avoir la forme réflechie de chaque mot

    if(strlen( Str: temp) > 1){ //si sa longeur est supérieure à 1, on le stocke

        fputs( Str: buf, File: in);
    }
```

On a constaté que le dictionnaire n'est pas complet des fois, il y a des lettres de l'alphabet qui ne sont pas intéressant pour construire une phrases, c'est pour on s'est permis de nettoyer le dictionnaire.

Par exemple tous les mots de base, la longeur est supérieure à 1, dans le cas on peut éliminer tous les alphabets

- Des exemples d'exécution sont fournis

```
Tapez 1 pour le modèle n°1 : nom – adjectif – verbe – nom
Tapez 2 pour le modèle n°2 : nom – 'qui' – verbe – verbe – nom – adjectif
Tapez 2 pour le modèle n°2 : nom – verbe – verbe – nom – adjectif
Tapez 0 pour quitter
Choisissez un modèle de phrase:
1
--> les vers futés kilomètrent les aqueducs
    (Base: ver futé kilométrer aqueduc)

Tapez 1 pour le modèle n°1 : nom – adjectif – verbe – nom
Tapez 2 pour le modèle n°2 : nom – 'qui' – verbe – verbe – nom – adjectif
Tapez 2 pour le modèle n°2 : nom – verbe – verbe – nom – adjectif
Tapez 0 pour quitter
Choisissez un modèle de phrase:
2
--> les won qui ulcérassiez vaincrez l'ibère annonciateur
    (Base: won qui ulcérer vaincre ibère annonciateur)

Tapez 1 pour le modèle n°1 : nom – adjectif – verbe – nom
Tapez 2 pour le modèle n°2 : nom – 'qui' – verbe – verbe – nom – adjectif
Tapez 2 pour le modèle n°2 : nom – verbe – verbe – nom – adjectif
Tapez 0 pour quitter
Choisissez un modèle de phrase:
```

```
Tapez 1 pour le modèle n°1 : nom – adjectif – verbe – nom
Tapez 2 pour le modèle n°2 : nom – 'qui' – verbe – verbe – nom – adjectif
Tapez 2 pour le modèle n°2 : nom – verbe – verbe – nom – adjectif
Tapez 0 pour quitter
Choisissez un modèle de phrase:
1
--> les pe dualistes ont warrantées le ïambe
    (Base: p dualiste warranter ïambe)

Tapez 1 pour le modèle n°1 : nom – adjectif – verbe – nom
Tapez 2 pour le modèle n°2 : nom – 'qui' – verbe – verbe – nom – adjectif
Tapez 2 pour le modèle n°2 : nom – verbe – verbe – nom – adjectif
Tapez 0 pour quitter
Choisissez un modèle de phrase:
2
--> le nom qui azure a fulguré la byssinose kolkhozien
    (Base: nom qui azurer fulgurer byssinose kolkhozien)

Tapez 1 pour le modèle n°1 : nom – adjectif – verbe – nom
Tapez 2 pour le modèle n°2 : nom – 'qui' – verbe – verbe – nom – adjectif
Tapez 2 pour le modèle n°2 : nom – verbe – verbe – nom – adjectif
Tapez 0 pour quitter
Choisissez un modèle de phrase:
```

- Liste des missions réalisées

- Accéder aux mots du dictionnaire

- Découper chaque ligne du fichier pour en extraire les informations

- Construire les arbres pour les Noms, les Verbes, les Adjectifs, les Adverbes

- Proposer une recherche de mot, vous rechercherez parmi les formes de base

- Extraire une forme de base au hasard

- Générerer des phrases au hasard selon plusieurs modèles

-Nettoyage du dictionnaire