MINI PROJET COMPRESSION
Algorithme de compression par déplacement en tête de liste

Introduction

On souhaite implanter une méthode d'encodage et de compression de données (adaptée aux textes) qui utilise une liste chaînée de mots. Dans la suite, la position d'un mot dans la liste est un entier strictement positif (la position du mot en tête de liste est donc 1).
Un mot est une suite de moins de 128 caractères. On ne demande pas de gérer le cas où un mot plus long apparaît.

Principe

Pour chaque mot du texte, si le mot existe dans la liste, on le code à l'aide de sa position dans la liste, puis on le déplace en tête de liste. Si le mot n'est pas dans la liste, on le code par l'entier 0 suivit de la taille du mot et du mot en clair, et on le place en fin de liste.
L'avancement du projet amènera à écrire les positions, les tailles et les mots sous forme d'octet.

Développement

Compression

Pour représenter une cellule, on a définit une structure _cell contenant une chaîne de caractère qui représentera le mot et un pointeur sur la cellule suivante :

```
typedef struct _cell {
  char *word;               /* chaîne de caractères. */
  struct _cell *next;       /* pointeur sur la cellule suivante. */
} cell;
```

Pour créer une cellule, on a définit une fonction allocate_cell qui prend en paramètre une chaîne de caractères constante :

```
cell *allocate_cell(const char *word){
    cell * cellule = (cell*) malloc(sizeof(cell)); //on alloue une cellule
    if(!cellule) //si la cellule est pas vide, on renvoie NULL
        return NULL;
    //La cellule a été allouée
    cellule->word = (char*)malloc(sizeof(char)*(strlen(word)+1));
    /*On alloue une chaîne de caractère de la taille
    du mot word +1 pour avoir assez de place pou le caractère \0*/
    if(!cellule->word)//si le mot n'a pas été alloué on s'arrete la
        return NULL;
```

```
    strcpy(cellule->word, word); //On copie la valeur de word dans la chaîne de
la cellule
    cellule->next = NULL; //La cellule n'est toujours pas attaché à la liste
donc on met son suivant à NULL
    return cellule;
}
```

Pour effectuer la compression, on exécute la méthode coding qui prend en paramètre le nom du fichier qui va être compressé et le nom de fichier qui va être le résultat de la compression.

```
void coding(char* entrer,char* sortie){
    FILE* file = fopen(entrer,"r"); //On ouvre le fichier à compresser en
lecture
    FILE* output = fopen(sortie,"wb"); //On créée un fichier que l'on met à
taille 0
    //qui va être le résultat de la compression du ficher en lecture
    int i=0;
        int j=0;
        char word[255];
        cell* lst = NULL;
    cell* ponct= NULL;
    int c = fgetc(file); //On lit le premier caractère du fichier

    //On va continuer de lire le fichier caractère par caractère,
    // tant que le curseur n'arrive pas à la fin du fichier (End Of File)
    while(c!= EOF){

        //tant que c est un caractère alphabétique, on continue de lire
        // la suite des caractères jusqu'à arriver à un mot
        while(isPonctuation(c)){
                        word[i] = c;
                        i++;
                        c = fgetc(file);
                }
        //si la variable i est > 0,cela veut dire qu'on a eu une suite de
caractères
        //on termine le mot avec \0 afin de dire que la variable n'a pas 255
caractères
        // On récupère le mot formé et on le met dans la liste. i est
réinitialisé à 0
                if(i){
                        word[i]= '\0';
                        add_word(&lst,output,word);
                        i = 0;
                        }

        //A ce moment c n'est pas alphanumérique, on continue la lecture char
par char jusqu'a tomber sur le prochain mot
        while(!isPonctuation(c) && c!= EOF){
                        word[j] = c;
                        j++;
                        c = fgetc(file);
                }
        //si la variable j est > 0,cela veut dire qu'on a eu une suite de
caractères de ponctuation
        //on termine le mot avec \0 afin de dire que la variable n'a pas 255
```

```
caractères
        // On récupère la ponctuation formée et on la met dans la liste dédiée à
la ponctuation. j est réinitialisé à 0
                if(j){
                                word[j]= '\0';
                                add_word(&ponct,output,word);
                                j=0;
                        }

        }
    free_cell(&lst); //On libère la liste
    free_cell(&ponct); // On libère la liste contenant la ponctuation
    fclose(output); //On ferme le fichier compressé
    fclose(file); //On ferme le fichier qui a servi à la compression
}
```

La fonction add_word prend en paramètre la liste chaînée qui contient les mots, le fichier qui va représenté la compression et le mot qui va être traité.

```
void add_word(cell* *lst,FILE* file, char word[]){
        cell* tmp = *lst;
    short pos = find_word((*lst),word); //On récupère la position du mot
    char len = strlen(word); //On récupère la taille du mot
    cell* found_word = NULL;

    if(*lst == NULL){ // SI la liste est vide, on alloue la première cellule de
la liste
                (*lst) = allocate_cell(word);
        //Dans le fichier qui va être le résultat de la compression :
        fwrite(&pos,sizeof(short),1,file); // On écrit la position du mot sous
forme d'octets
        fwrite(&len,sizeof(char),1,file); // On écrit la taille du mot sous
forme d'octets
        fwrite(word,sizeof(char),len,file); // On écrit le mot sous forme
d'octets
        return; //On termine l'éxècution de la méthode
        }

        if(pos == 0){
        cell * new_cell = allocate_cell(word); //SI le mot n'a pas été trouvé
dans la liste on va l'inséré en fin de liste
                while (tmp->next != NULL){
            tmp = tmp->next;
                }
        //Dans le fichier qui va être le résultat de la compression :
        fwrite(&pos,sizeof(short),1,file); // On écrit la position du mot sous
forme d'octets
        fwrite(&len,sizeof(char),1,file); // On écrit la taille du mot sous
forme d'octets
        fwrite(word,sizeof(char),len,file); // On écrit le mot sous forme
d'octets
        tmp->next = new_cell; //On insère la cellule créée en fin de liste
        return;// On termine l'éxècution de la méthode
        }

    //Si la liste n'est pas vide et que le mot existe dans la liste
```

```
    //On écrit la position du mot dans la liste, sa position nous servira à le
retrouver
        fwrite(&pos,sizeof(short),1,file);
    //found_word = extract_cell(lst,word);
        if(pos > 1){
        found_word = extract_cell(lst,word); // Si il y a plus d'une occurences
dans la liste en plus de du mot
        // On met la cellule en tête de liste et on attache la cellule à la
liste
        (*lst) = found_word; //
        (*lst)->next = tmp;//
        tmp = NULL;//
        }
}
```

Décompression

Pour effectuer la décompression, on exécute la méthode uncoding qui prend en paramètre le nom du fichier est compressé et le nom de fichier qui va être le résultat de la décompression.

```
void uncoding(char* entrer,char* sortie){
    FILE* file = fopen(entrer,"rb"); //On ouvre le fichier à décompresser en
lecture
    FILE* result = fopen(sortie,"w"); //On créée un fichier que l'on met en
écriture
    //Les variables suivantes vont servir au traitement du fichier à
décompresser
    cell* lst = NULL;
        cell* ponct = NULL;
        char word[255];
        cell* tmplst = lst;
        cell* tmponct = ponct;
        cell* found_word= NULL;
        char size;
        short c;
        int type,i;

    //on lit le fichier short par short du fichier compréssé tant que on arrive
pas à la fin du fichier
        while(fread(&c,sizeof(short),1,file)){
         //Si le short est égale à 0, cela signifie que l'on a trouvé un nouveau
mot
                if(c == 0){
            fread(&size,sizeof(char),1,file);   //On lit la taille du mot
            fread(&word,sizeof(char),size, file);       // On le mot en entier
grâce à sa taille
            word[size] = '\0'; // On délimite le mot par \0 pour montrer le
tableau n'a pas 255 caractères
            fprintf(result, "%s",word); // On écrit le mot dans le fichier de
décompression
            found_word = allocate_cell(word); // On créée une cellule avec le
mot écrit précédemment
            tmplst = lst; //On stock temporairement la tête de liste contenant
les mots
```

```
            tmponct = ponct; // On stock temporairement la tête de liste
contenant la ponctuation du texte
            //Si le premier élément du mot est alphanumérique, le mot est un mot
contenant des lettres ou des chiffres
            if((word[0] >= 'a' && word[0] <= 'z') || (word[0] >= 'A' &&
word[0]<= 'Z') || (word[0] >= '0' && word[0]<= '9') ){
                                if(lst == NULL)
                    lst = found_word; //Si la liste est vide, on initialise la
première cellule par celle allouée précédemment
                                else{
                    while (tmplst->next != NULL) //Si la liste n'est pas vide,
on met la cellule allouée précédemment en fin de liste
                                                tmplst = tmplst->next;
                                            tmplst->next = found_word;
                                }
                type=0; //On met le type à 0 pour signifier que le mot est bien
alphanumérique
                            }
            //SI le premier élément n'est pas alphanumérique
                            else{
                                            if(ponct == NULL)
                    ponct = found_word;  //Si la liste de ponctuation est vide,
on initialise la première cellule par celle allouée précédemment
                                else{
                    while (tmponct->next != NULL) //Si la liste de ponctuation
n'est pas vide, on met la cellule allouée précédemment en fin de liste
                                                tmponct = tmponct->next;
                                            tmponct->next = found_word;
                                }
                type=1; //On met le type à 1 pour signifier que le mot est bien
de la ponctuation
                            }
                }

        // Si c n'est pas égale à 0, on récupère une position qui représente le
mot dans la liste de mots ou de ponctuation.
                else{
            //Si le mot précédent est non alphanumérique
                            if(type == 1){
                tmplst = lst; // On stocke la tête de liste

                                for (i = 0; i < c-1; i++)
                    tmplst = tmplst->next; //On récupère la cellule de la liste
de mots à la position du mot
                fprintf(result, "%s",tmplst->word); //On écrit le mot dans le
fichier de décompression
                found_word = extract_cell(&lst,tmplst->word); //On extrait la
cellule contenant le mot et on la stocke found_word
                tmplst = lst; //On stocke la tête de liste
                lst = found_word; // On met la cellule trouvée précédemment en
tête de liste
                lst->next = tmplst; // On attache le reste de la liste chaînée
contenant les mots.
                type = 0; //On précise que après un mot il y a forcément une
ponctuation.
                            }
            else{ // Si le mot précédent est alphanumérique

                tmponct = ponct; //On stocke la tête de liste de ponctuation
```

```
                                for (i = 0; i < c-1; i++)
                        tmponct = tmponct->next;  //On récupère la cellule de la
liste de mots à la position du mot
                        fprintf(result, "%s",tmponct->word);//On écrit la ponctuation
dans le fichier de décompression
                        found_word = extract_cell(&ponct,tmponct->word); //On extrait la
cellule contenant la ponctuation et on la stocke found_word
                        tmponct = ponct; //On stocke la tête de liste de ponctuation
                        ponct = found_word; // On met la cellule trouvée précédemment en
tête de liste de ponctuation
                        ponct->next = tmponct;  // On attache le reste de la liste
chaînée contenant la ponctuation.
                        type = 1;//On précise que après une ponctuation il y a forcément
un suite alphanumérique.
                                }
                        }
                }

    free_cell(&lst); //On libère la liste contenant les mots
    free_cell(&ponct); // On libère la liste contenant la ponctuation
    fclose(file); //On ferme le fichier compressé
    fclose(result);//On ferme le fichier décompressé
}
```

Conclusion

Ce mini projet nous a permis d'explorer les possibilités du langage C sur la compression des fichiers en binaires. Nous avons su rendre un fichier texte complètement illisible pour un utilisateur lambda. Et le rendre lisible après décompression. Cela nous a appris comment rendre un fichier plus léger et nous a permis de nous perfectionner avec les listes chaînées et les entrées/sorties des fichiers en C.