

# TD9 – Manipulation de fichiers

Langage C (LC4)

Semaine du 2 avril 2012

## 1 Découpage d'un fichier mot par mot

Vous pourrez utiliser `int fscanf(FILE *stream, const char *format, ...)` avec le format `%s`, qui récupère une chaîne de caractères jusqu'à un espace ou un saut de ligne, et avec l'option `a` qui alloue la mémoire nécessaire pour stocker la chaîne trouvée. La valeur renvoyée est le nombre de motifs trouvés.

**Question 1.** Écrivez une fonction `char *prochain_mot(FILE *f)` qui renvoie la chaîne de caractères contenue entre la position courante dans `f` et le premier espace ou saut de ligne. Renvoyez `NULL` à la fin du fichier.

**Solution.**

```
char *prochain_mot(FILE *f)
{
    char *buffer;
    if (fscanf(f, "%as", &buffer) > 0)
        return buffer;
    else
        return NULL;
}
```

**Question 2.** Écrivez une fonction `void decoupe_et_reconstruit(FILE *f1, FILE *f2)` qui lit le fichier texte de `f1` et l'écrit dans `f2` en mettant un mot par ligne.

**Solution.**

```
void decoupe_et_reconstruit(FILE *f1, FILE *f2)
{
    char *mot;
    while (mot = prochain_mot(f1))
        fprintf(f2, "%s\n", mot);
}
```

**Question 3.** Écrivez une fonction `void decoupe_et_affiche(FILE *f1)` qui lit le fichier texte de `f1` et l'écrit sur la sortie standard en mettant un mot par ligne.

**Solution.**

```
void decoupe_et_affiche(FILE *f1)
{
    decoupe_et_reconstruit(f1, stdout);
}
```

**Question 4.** Écrivez une fonction `int main(int argc, char *argv[])` qui appelle la fonction `decoupe_et_reconstruit` sur deux fichiers dont les noms sont passés en argument au programme.

**Solution.**

```
int main(int argc, char *argv[])
{
    if (argc != 3) {
        printf("Entrez 2 noms de fichiers svp!\n");
        return EXIT_FAILURE;
    }
    FILE *f1 = fopen(argv[1], "r");
    if (f1 == NULL) {
        perror("Problème 'a l'ouverture de argv[1]");
        return EXIT_FAILURE;
    }
    FILE *f2 = fopen(argv[2], "w");
    if (f2 == NULL) {
        perror("Problème 'a l'ouverture de argv[2]");
        return EXIT_FAILURE;
    }
    decoupe_et_reconstruit(f1, f2);
    fclose(f1);
    fclose(f2);
    return EXIT_SUCCESS;
}
```

## 2 Découpage d'un fichier mot par mot et stockage dans un tableau

**Question 5.** Écrivez une fonction `char **decoupe_en_tableau(FILE *f)` qui lit le fichier texte passé en argument et renvoie un tableau de chaînes de caractères dont chaque case contient un mot du texte. La dernière case du tableau contiendra `NULL`.

**Solution.**

```
char **decoupe_en_tableau(FILE *f)
{
    char **tab = malloc(sizeof(char*));
    if (tab == NULL) {
        perror("L'allocation a 'echou'e");
        exit(EXIT_FAILURE);
    }
    int capacite = 1;
    int occupation = 0;
    while (tab[occupation++] = prochain_mot(f)) {
        if (occupation == capacite) {
            capacite *= 2;
            tab = realloc(tab, capacite * sizeof(char*));
            if (tab == NULL) {
                perror("La r'eallocation a 'echou'e");
                exit(EXIT_FAILURE);
            }
        }
    }
}
```

```

    }
  }
}
return tab;
}

```

**Question 6.** Écrivez une fonction `void reconstruit_depuis_tableau(char **tab, FILE *f)` qui écrit dans `f` les mots contenus dans `tab` séparés par des tabulations. On suppose que le dernier élément de `tab` est `NULL`.

**Solution.**

```

void reconstruit_depuis_tableau(char **tab, FILE *f)
{
    while (*tab != NULL) {
        fprintf(f, "%s\t", *tab);
        tab++;
    }
    printf("\n");
}

```

### 3 Déplacement dans un fichier

Les fonctions utilisées jusqu'ici parcourent un fichier séquentiellement du début à la fin, en déplaçant implicitement un curseur dans le fichier. On peut jouer avec ce curseur grâce à la fonction `int fseek(FILE *f, long decalage, int origine)`. La valeur renvoyée est 0 en cas de succès, autre chose sinon. Cette fonction permet de se positionner en `origine + decalage`. Le paramètre `decalage` est exprimé en octets et `origine` peut prendre les 3 valeurs suivantes :

`SEEK_SET` : début du fichier ;

`SEEK_CUR` : position courante ;

`SEEK_END` : fin du fichier ;

Pour connaître la position courante à partir du début du fichier, on utilise `long ftell(FILE *f)`.

**Question 7.** Écrivez une fonction `void moities(FILE *f1, FILE *f2, FILE *f3)` qui écrit la première moitié de `f1` dans `f2` et la seconde moitié de `f1` dans `f3`.

**Solution.**

```

void moities(FILE *f1, FILE *f2, FILE *f3)
{
    fseek(f1, -1, SEEK_END);
    long taille = ftell(f1);
    fseek(f1, 0, SEEK_SET);
    long i = 0;
    int c;
    while (i++ <= taille / 2) {
        c = fgetc(f1);
        fputc(c, f2);
    }
}

```

```

    }
    while (i++ <= taille) {
        c = fgetc(f1);
        fputc(c, f3);
    }
}

```

**Question 8.** Écrivez une fonction `int main(int argc, char *argv[])` qui appelle la fonction `moities` sur trois fichiers dont les noms sont passés en argument au programme.

**Solution.**

```

int main(int argc, char *argv[])
{
    FILE *f1 = fopen(argv[1], "r");
    FILE *f2 = fopen(argv[2], "w");
    FILE *f3 = fopen(argv[3], "w");
    moities(f1, f2, f3);
    fclose(f1), fclose(f2), fclose(f3);
    return 0;
}

```

**Question 9.** Écrivez une fonction `void renverse(FILE *f1, FILE *f2)` qui lit le texte contenu dans `f1` et l'affiche à l'envers caractère par caractère sur `f2`.

**Solution.**

```

void renverse(FILE *f1, FILE *f2)
{
    char c;
    fseek(f1, 0, SEEK_END);
    while (fseek(f1, -1, SEEK_CUR) == 0) {
        c = fgetc(f1);
        fputc(c, f2);
        fseek(f1, -1, SEEK_CUR);
    }
}

```

On définit un type liste de caractères ainsi :

```

typedef struct carac *buffer;
struct carac {
    char val;
    buffer suivant;
};

```

On utilisera cette structure de données comme un tampon du genre LIFO (last in first out).

**Question 10.** Écrivez une fonction `buffer ajouter(char c, buffer b)` qui ajoute `c` en tête de `b` et renvoie un pointeur vers la tête du buffer.

**Solution.**

```

buffer ajouter(char c, buffer b)
{

```

```

buffer tete = malloc(sizeof(*tete));
tete -> val = c;
tete -> suivant = b;
return tete;
}

```

**Question 11.** Écrivez une fonction récursive **void** `ecrire(buffer b, FILE *f)` qui écrit le contenu de `b` dans `f` et libère la mémoire occupée par le buffer `b`.

**Solution.**

```

void écrire(buffer b, FILE *f)
{
    buffer queue;
    if (b != NULL) {
        fputc(b->val, f);
        queue = b->suivant;
        free(b);
        écrire(queue, f);
    }
}

```

**Question 12.** En utilisant `buffer`, écrivez une fonction **void** `renverse_lignes(FILE *f1, FILE *f2)` qui écrit dans `f2` les lignes de `f1` en partant de la dernière pour finir par la première.

**Solution.**

```

void renverse_lignes_buffer (FILE *f1, FILE *f2)
{
    buffer b = NULL;
    int c;
    fseek(f1, -1, SEEK_END);
    while (1) {
        while ((c = fgetc(f1)) != '\n') {
            b = ajouter(c, b);
            if (ftell(f1) == 1)
                break;
            else
                fseek(f1, -2, SEEK_CUR);
        }
        écrire(b, f2);
        fputc('\n', f2);
        b = NULL;
        if (ftell(f1) >= 2)
            fseek(f1, -2, SEEK_CUR);
        else
            break;
    }
}

```

**Question 13.** Même question, mais sans utiliser `buffer`. Pour attraper une ligne, vous pourrez utiliser `fscanf` avec le format `%a[^\n]`; on rappelle que le `a` vous dispense de l'allocation de mémoire.

### Solution.

```
void renverse_lignes (FILE *f1, FILE *f2)
{
    char *ligne;
    fseek(f1, -1, SEEK_END);
    while (1) {
        int longueur = 0;
        while (fgetc(f1) != '\n') {
            longueur++;
            if (ftell(f1) == 1) {
                fseek(f1, -1, SEEK_CUR);
                break;
            } else {
                fseek(f1, -2, SEEK_CUR);
            }
        }
        fscanf(f1, "%a[^\n]", &ligne);
        fprintf(f2, "%s\n", ligne);
        fseek(f1, -longueur, SEEK_CUR);
        if (ftell(f1) >= 2)
            fseek(f1, -2, SEEK_CUR);
        else
            return;
    }
}
```

**Question 14.** En utilisant buffer, écrivez une fonction **void** droite\_gauche(FILE \*f1, FILE \*f2) qui réécrit dans f2 les lignes de f1 dans l'autre sens.

### Solution.

```
void droite_gauche_buffer(FILE *f1, FILE *f2)
{
    int c;
    int d;
    buffer b = NULL;
    while (1) {
        while ((c = fgetc(f1)) != '\n' && c != EOF) {
            b = ajouter(c, b);
        }
        ecrire(b, f2);
        fputc('\n', f2);
        b = NULL;
        if (c == EOF)
            return;
    }
}
```

**Question 15.** Même question, mais sans utiliser buffer.

### Solution.

```
void droite_gauche(FILE *f1, FILE *f2)
{
```

```

int c;
char d;
while (1) {
    do {
        c = fgetc(f1);
    } while (c != '\n' && c != EOF);
    int longueur = 0;
    do {
        fseek(f1, -2, SEEK_CUR);
        d = fgetc(f1);
        fputc(d, f2);
        longueur++;
    } while (d != '\n' && ftell(f1) != 1);
    if (ftell(f1) == 1)
        fputc('\n', f2);
    if (c == EOF)
        return;
    else
        fseek(f1, longueur + 1, SEEK_CUR);
}
}

```

**Question 16.** Discutez des avantages et des inconvénients des 2 méthodes (avec ou sans buffer).

**Solution.** C'est plus facile à écrire avec buffer car on fait moins d'allers-retours dans le fichier, mais avec buffer, on utilise 1 octet sur 5, ce qui n'est pas très efficace. Il n'y a pas beaucoup d'avantages à utiliser buffer à part réviser les structures et l'allocation dynamique.