

# Fichiers et tokenization

Inspiré des slides de F.Gluck et O.Malaspinas

---

13-12-2022

# Les fichier en C

- Un fichier en C est représenté par un pointeur de fichier.

```
#include <stdio.h>  
FILE *fp;
```

- `FILE *` est une structure de donnée *opaque* contenant les informations sur l'état du fichier.
- Il est manipulé à l'aide de fonctions dédiées.
- Le pointeur de fichier est toujours passé *par référence*.

# Manipulations de fichier

- Pour lire/écrire dans un fichier il faut toujours effectuer trois étapes:
  1. Ouvrir le fichier (`fopen()`).
  2. Écrire/lire dans le fichier (`fgets()`, `fputs()`, `fread()`, `fwrite()`, ...).
  3. Fermer le fichier (`fclose()`).
- Nous allons voir brièvement ces trois étapes.

# Ouverture d'un fichier

- L'ouverture d'un fichier se fait avec la fonction `fopen()`

```
FILE *fp = fopen(filename, mode);
```

- `filename` est une chaîne de caractères (incluant le chemin).
- `mode` est le mode d'ouverture ("`r`", "`w`", ... voir man 3 `fopen`) qui est une chaîne de caractères.
- Si l'ouverture échoue (pas la bonne permission, ou n'existe pas) `fopen()` retourne 0.
- **Toujours** tester le retour de `fopen()`

```
if (NULL == fp) {  
    fprintf(stderr, "Can't open output file %s!\n",  
            filename); // affiche dans le canal d'erreur  
    exit(EXIT_FAILURE);  
}
```

# Fermeture d'un fichier

- Il faut **toujours** fermer un fichier à l'aide de `fclose()`

```
FILE *fp = fopen("mon_fichier", "w");  
// écritures diverses  
fclose(fp);
```

- Les données sont transférées dans une mémoire tampon, puis dans le disque.
- Si la mémoire tampon est pleine, le fichier est fermé, ... les données sont écrites sur le disque.
- Si la mémoire tampon contient encore des données à la fin du programme les données sont **perdues**.

# Lecture d'un fichier

```
char *fgets(char *s, int size, FILE *stream)
```

- Lit une ligne de taille d'au plus `size-1` et la stocke dans `s`, depuis `stream`.
- Retourne `s` ou `NULL` si échoue.

```
FILE *fp = fopen("text.txt", "r");  
char buffer[100];  
fgets(buffer, 37, fp);  
// lit 36 caractères au plus et les écrit dans buffer  
// s'arrête si rencontre EOF, ou "\n".  
// ajoute un "\0" terminal  
fclose(fp);
```

# Lecture d'un fichier

- Exemple de lecture de toutes les lignes d'un fichier

```
while(NULL != fgets(buf, BUFSIZE, fin)){  
    printf("%s", buf);  
}
```

# Lecture d'un fichier

```
size_t fread(void *ptr, size_t size, size_t nmemb,  
             FILE *stream)
```

- Lit nmemb éléments de taille size octets et les écrit à l'adresse ptr depuis le fichier stream.
- Retourne le nombre d'éléments lus.

```
FILE *fp = fopen("doubles.bin", "rb");  
double buffer[100];  
size_t num = fread(buffer, sizeof(double), 4, fp);  
// lit 4 double, se trouvant à l'adresse  
// buffer dans le fichier fp au format binaire  
// retourne 4  
fclose(fp);
```



# Écriture dans un fichier

```
int fprintf(FILE *stream, const char *format, ...)
```

- Écrit la chaîne de caractères format dans le fichier stream.
- format a la même syntaxe que pour printf().
- Retourne le nombre de caractères écrit sans le \0 terminal (si réussite).

```
FILE *fp = fopen("text.txt", "w");  
fprintf(fp, "Hello world! We are in %d\n", 2020);  
// Écrit "Hello world! We are in 2020"  
// dans le fichier fp  
fclose(fp);
```

# Écriture dans un fichier

```
size_t fwrite(const void *ptr, size_t size,  
              size_t nmemb, FILE *stream)
```

- Écrit nmemb éléments de taille size octets se trouvant à l'adresse ptr dans le fichier stream.
- Retourne le nombre d'éléments écrits.

```
FILE *fp = fopen("doubles.bin", "wb");  
double buffer[] = {1.0, 2.0, 3.0, 7.0};  
size_t num = fwrite(buffer, sizeof(double), 4, fp);  
// écrit 4 double, se trouvant à l'adresse  
// buffer dans le fichier fp au format binaire  
// retourne 4  
fclose(fp);
```

# Les pointeurs de fichiers spéciaux

- Il existe trois `FILE *` qui existent pour tout programme:
  - `stdin`: l'entrée standard.
  - `stdout`: la sortie standard.
  - `stderr`: l'erreur standard.
- Lors d'un fonctionnement dans le terminal:
  - l'entrée standard est le *clavier*
  - la sortie et erreur standard sont affichés dans le *terminal*.
- Ainsi on a

```
fprintf(stdout, "texte\n"); // == printf("texte\n");  
int a;  
fscanf(stdin, "%d", &a); // == scanf("%d", &a);
```

# Tokenisation

## Tokenisation : séparer une chaîne de caractères en parties (tokens)

- Utilisation d'une fonction de tokenization
- Définir les caractères séparateurs

### En C, fonction strtok()

```
char * strtok ( char * str, const char * delimiters );
```

- A chaque appel, retourne un pointeur sur le prochain token;
- Au premier appel, passer la chaîne à tokeniser comme premier paramètre;
- Aux appels suivants, passer NULL comme premier paramètre;
- Retourne NULL lorsque plus de token trouvé.

### Attention ! strtok() modifie la chaîne passée au premier appel

- Ajout des caractères \0 à la place des séparateurs.

# Tokenisation

## Exemple d'utilisation de strtok()

```
#include <string.h>

char str[] = "salut les amis";
char* tok = strtok(str, " ");
while(NULL != tok){
    printf("%s\n", tok);
    tok = strtok(NULL, " ");
}
```