

UNIVERSITÉ CHEIKH ANTA DIOP DE DAKAR
FACULTÉ DES SCIENCES ET TECHNIQUE
DÉPARTEMENT MATHÉMATIQUES ET INFORMATIQUE

Chapitre 7 – LES FICHIERS EN C

Cours de programmation en LANGUAGE C
Licence 1 Informatique

2019-2020

Dr. Ibrahima DIANÉ

Introduction

- Les données d'un programme stockées en mémoire centrale sont perdues dès la fin de l'exécution du programme.
- Le langage C offre la possibilité de lire et d'écrire des données dans un fichier.
- Mais, avant de lire ou d'écrire dans un fichier, celui-ci doit être ouvert.
- Après les traitements, le fichier doit être fermé.

Notion de flux

- Dans un programme en langage C, on manipule un fichier grâce à ce qu'on appelle un flux . Ce flux est associé au fichier au moment de son ouverture.
- Il existe trois flux standards qui sont définis dans stdio.h :
 - **stdin** (flux d'entrée standard) : est associé **au clavier** et permet de lire ce qui est entré dans ce périphérique d'entrée.
 - **stdout** (flux de sortie standard) : est associé **à l'écran** et permet d'afficher des informations lors du déroulement normal du programme.
 - **stderr** (flux de sortie standard d'erreur) : est associé **à l'écran** et permet d'afficher des messages d'erreurs lors de l'exécution du programme.
- En C, un flux est de type FILE*
 - Le type FILE ainsi que l'ensemble des fonctions qui permettent de faire des manipulations sur les fichiers sont définies dans la bibliothèque standard stdio.h.

Ouverture d'un fichier

- La fonction **fopen** permet d'ouvrir un fichier.
 - Son prototype est le suivant : `FILE* fopen(char*, char*);`
 - La première chaîne est le nom du fichier (nom complet ou nom relatif).
 - La seconde chaîne est le mode d'ouverture (lecture, écriture ou ajout de données).
 - Elle retourne un flux (une valeur de type `FILE*`)
 - Si l'ouverture échoue pour une raison quelconque , **fopen** retourne la valeur `NULL` .
- Ainsi, pour manipuler un fichier dans un programme, il faut d'abord l'ouvrir avec la fonction **fopen** en affectant la valeur retournée à un flux préalablement déclaré et c'est ce dernier qui va être utilisé pour toutes les autres opérations sur le fichier.

Ouverture d'un fichier

- **Syntaxe d'ouverture d'un fichier:**

```
FILE* nomFlux;  
nomFlux = fopen(nom_fichier, mode_d_ouverture);
```

- Si l'ouverture se déroule normalement, c'est la variable `nomFlux` qui sera utilisée par la suite pour effectuer toutes les opérations permises par le mode d'ouverture qu'on a choisi.
- **Modes d'ouverture d'un fichier**

| mode | signification |
|------|---|
| "r" | Ouvre un fichier existant pour la lecture |
| "w" | Crée et ouvre un fichier pour l'écriture. Si le fichier existe déjà, il sera écrasé. |
| "a" | Ouvre un fichier en ajout. L'ajout se fait à la fin d'un fichier existant. Si le fichier n'existe pas, il sera créé. |
| "r+" | Ouvre un fichier existant pour la lecture et l'écriture |
| "w+" | Crée et ouvre un fichier pour la lecture et l'écriture. Si le fichier existe déjà, il sera écrasé |
| "a+" | Ouvre un fichier pour la lecture et l'ajout de données à la fin d'un fichier existant. Si le fichier n'existe pas, il sera créé. |

Ouverture d'un fichier

- Il est recommandé de toujours tester la valeur de retour de la fonction **fopen** afin de détecter une éventuelle erreur d'ouverture.
- **Exemple**

```
#include <stdio.h>

int main()
{
    FILE*fp;
    fp = fopen ("c:\\algo\\essai.txt", "r");

    if (fp == NULL)
    {
        printf("Impossible d'ouvrir le fichier");
        return 0;
    }
    ...
}
```

Fermeture d'un fichier

- Tout fichier ouvert doit être fermé après son utilisation avec **fclose()** selon la syntaxe suivante :

fclose (nomFlux) ;

- **Exemple :**

```
FILE * fp;  
fp = fopen ("C:\\algo\\essai.txt", "w");  
...  
fclose(fp);
```

Lecture et écriture dans un fichier texte

Lecture et écriture formatés

- La fonction **fprintf** permet d'écrire dans un fichier. Elle s'utilise exactement de la même manière que **printf** mais en ajoutant le flux comme premier paramètre.
- Sa syntaxe est la suivante :

```
fprintf(nomFlux, " chaîne de caractères", expression1, ...,  
expressionN) ;
```

- La fonction **fscanf** permet de lire à partir d'un fichier. Elle s'utilise exactement de la même manière que **fscanf** mais en ajoutant le flux comme premier paramètre.
- Sa syntaxe est la suivante :

```
fscanf(nomFlux, " chaîne de formatage", &variable1, ...,  
&variableN) ;
```


Lecture et écriture dans un fichier texte

Lecture et écriture de caractère

- Similaires aux fonctions `getchar` et `putchar`, les fonctions `fgetc` et `fputc` permettent respectivement de lire et d'écrire un caractère dans un fichier.
- **Prototypes :**

```
int fgetc(FILE* fp);  
int fputc(int caractere, FILE *fp);
```
- La fonction **`fgetc()`** retourne le code ASCII du caractère lu à partir du fichier, ou **EOF** en cas de lecture de la fin de fichier ou en cas d'erreur.
- La fonction **`fputc()`** retourne le code ASCII du caractère écrit dans le fichier, ou **EOF** en cas d'erreur.

Exemple d'utilisation de `fgetc`

```
char c;  
FILE* fp;  
...  
c = fgetc(fp);
```

Exemple d'utilisation de `fputc`

```
FILE* fp;  
...  
fputc('A', fp);
```


Lecture et écriture dans un fichier texte

- **Exemple**

Programme qui lit le contenu d'un fichier texte, et le recopie caractère par caractère dans un autre fichier:

```
#include <stdio.h>
#define ENTREE "c:\\algo\\essai.txt"
#define SORTIE "c:\\algo\\essai2.txt"

int main()
{
    FILE *fp_in, *fp_out;
    int c;
    if ((fp_in = fopen(ENTREE, "r")) == NULL)
    {
        printf("\nErreur: échec d'ouverture du fichier %s\n", ENTREE);
        getchar();
        return 1;
    }
    if ((fp_out = fopen(SORTIE, "w")) == NULL)
    {
        printf("\nErreur: échec d'ouverture du fichier %s\n", SORTIE);
        getchar();
        return 1;
    }
    ...
```



```
        c = fgetc(fp_in);
        while (c != EOF)    // tant qu'on n'a
            pas lu la fin du fichier
        {
            fputc(c, fp_out);
            c = fgetc(fp_in);
        }

        fclose(fp_in);
        fclose(fp_out);

        return 0;
    }
```

Lecture et écriture dans un fichier texte

Lecture de chaînes de caractères

- La fonction **fgets** permet la lecture d'une chaîne de caractères ch à partir d'un flux. Son prototype est :

```
char* fgets(char*, int , FILE *);
```

- **fgets** prend en paramètre la variable chaîne de caractères qui doit recevoir la chaîne lue, un nombre entier et un flux. La valeur retournée est un pointeur de type char *
- **Syntaxe d'utilisation** : `fgets(ch, n, nomFlux);`
- La lecture s'effectue jusqu'à **n-1** caractères au maximum. Si le caractère '\n' est rencontré avant d'atteindre n-1 caractères, alors '\n' est ajouté à ch et la lecture s'arrête. Ensuite le caractère '\0' est ajouté à ch.
- La fonction **fgets()** retourne la valeur **NULL** en cas d'erreur ou de détection de fin de fichier.
- **Exemple d'utilisation de fgets:**

```
char ch[50];  
FILE * fp;  
...  
fgets(ch, 50, fp);
```

Lecture et écriture dans un fichier texte

Écriture de chaînes de caractères

- La fonction **fputs** permet l'écriture d'une chaîne de caractères `ch` dans un fichier. Son prototype est :

```
int fputs(const char*ch, FILE*stream);
```

- **fputs** retourne le code ASCII du dernier caractère écrit, ou **EOF** en cas d'erreur.
- **Exemple d'utilisation de fputs:**

```
char ch[] = "Hello world";  
FILE * fp;  
...  
fputs( ch, fp);
```

NB

- Toutes les fonctions d'écriture qu'on vient de voir (`fprintf`, `fputc`, `fputs`) peuvent être utilisées pour écrire sur la sortie standard (l'écran). Dans ce cas, le flux est **stdout**.
- Toutes les fonctions de lecture qu'on vient de voir (`fscanf`, `fgetc`, `fgets`) peuvent être utilisées pour lire à partir de l'entrée standard (le clavier). Dans ce cas, le flux est **stdin**.

Lecture et écriture dans un fichier binaire

- Pour un fichier ouvert en mode binaire, les fonctions de lecture et d'écriture à utiliser sont respectivement **fread** et **fwrite**. Ces fonctions permettent de lire ou d'écrire des blocs d'octets.

- **Syntaxe de lecture avec fread :**

```
fread(adresse, taille_bloc, nb_bloc, nomFlux);
```

- Cette instruction permet de lire à partir du fichier un nombre **nb_bloc** de blocs dont chacun est de taille **taille_bloc** et les copie en mémoire à partir de l'adresse **adresse**.
- La fonction **fread** retourne le nombre de blocs lus. Une erreur de lecture a eu lieu si ce nombre est inférieur au nombre **nb_bloc** qu'on a indiqué dans l'instruction.

Lecture et écriture dans un fichier binaire

- **Syntaxe de lecture avec fwrite :**

```
fwrite(adresse, taille_bloc, nb_bloc, nomFlux);
```

- Ecrit dans le fichier un nombre **nb_bloc** de blocs dont chacun est de taille **taille_bloc** lus en mémoire à partir de l'adresse **adresse**.
- La fonction **fwrite** retourne le nombre de blocs écrits. Une erreur d'écriture a eu lieu si ce nombre est inférieur au nombre **nb_bloc** qu'on a indiqué dans l'instruction.

- **Exemple**

```
int tab[ ] = {2, 8, 6, -9, 7, 6, 78, 25, 0, 6};
```

```
FILE * fp;
```

```
...
```

```
fwrite(tab, sizeof(int), 10, fp); // tout le tableau est copié dans le  
fichier
```

```
/* ou bien: fwrite(tab, 10*sizeof(int), 1, fp); */
```

Positionnement dans un fichier

- L'écriture ou la lecture dans un fichier se fait suivant un curseur positionné à l'endroit où la dernière opération de lecture ou d'écriture a eu lieu.
- Il est possible de changer la position du curseur en utilisant les fonctions **fseek** et **rewind**.

La fonction **fseek**

- La fonction **fseek** permet de déplacer le curseur à un endroit précis du fichier. Sa syntaxe est:

```
fseek(nomFlux, déplacement, origine);
```
- L'argument `deplacement` représente le déplacement (positif ou négatif) en nombre d'octets à effectuer dans le fichier. Il s'agit d'un déplacement par rapport à l'argument `origine`. Il doit être de type **long**, donc si une constante est utilisée, elle doit avoir le suffixe **L** (par exemple 10L).
- L'argument `origine` peut prendre trois valeurs:
 - **SEEK_SET** (ou 0): début du fichier;
 - **SEEK_CUR** (ou 1): position courante;
 - **SEEK_END** (ou 2): fin du fichier.
- La fonction **fseek()** retourne la valeur 0 si le curseur a pu être déplacé.

Positionnement dans un fichier

La fonction **rewind**

- La fonction **rewind** permet de déplacer le curseur au début du fichier.
- Sa syntaxe est:

```
rewind(nomFlux);
```

- Elle est équivalente à:

```
fseek(nomFlux, 0, SEEK_SET);
```


Gestion des erreurs

Test de la valeur de retour des fonctions

- Pour gérer les erreurs éventuelles, il est plus rigoureux de tester les valeurs de retour des fonctions utilisées.
- **Exemple :** lecture des lignes d'un fichier avec fgets et affichage à l'écran

```
int main( )
{
int n = 0; /* nombre de lignes lues */
char ligne[256];
FILE *fp = fopen( "c:\\exemple.txt", "r" ); //ouvrir en lecture
if (fp==NULL)
{
printf("erreur d'ouverture\n");
getchar();
return 1; // arrêter le programme
}
while (fgets( ligne, 256, fp ) != NULL) //essai de lecture d'une ligne
{
n++;
printf("%s\n", ligne); // affichage de la ligne lue
}
fclose(fp); // fermer le fichier
printf("%d lignes lues\n", n); // affichage du nombre de lignes lues
getchar();
return 0;
}
```

Gestion des erreurs

Utilisation de la fonction feof

- La fonction **feof** retourne une valeur non nulle (VRAI au sens booléen) si et seulement si on fait une opération de lecture alors qu'on est à la fin du fichier. Dans tous les autres cas, feof retourne une valeur nulle (FAUX au sens booléen). Elle a pour prototype:

```
int feof(FILE *fp);
```

- Elle peut donc être appelée immédiatement après une lecture sur le fichier pour savoir si l'opération a échoué ou non.

Exemple d'utilisation

- Pour lire un fichier du début à la fin, on peut utiliser le schéma suivant:

```
FILE * fp;
...
ouverture du fichier
vérification du succès de l'ouverture du fichier
...
lecture sur le fichier
while(feof(fp)== 0)                // ou bien while(!feof(fp))
{
    traitement des données lues
    lecture sur le fichier
}
...
```

Gestion des erreurs

Utilisation de la fonction `ferror`

- La fonction **`ferror`** retourne une valeur **non nulle (VRAI au sens booléen)** si une erreur s'est produite lors d'une lecture ou une écriture sur un fichier. Elle retourne une valeur nulle (FAUX au sens booléen) si aucune erreur ne s'est produite.
- Elle a pour prototype :

```
int ferror(FILE *fp) ;
```

- Elle peut donc être appelée immédiatement après une lecture ou une écriture sur le fichier pour savoir si l'opération en question a échoué ou non.

FIN