Initiation à l'algorithmique
Les fichiers

Mohamed MESSABIHI

mohamed.messabihi@gmail.com

Université de Tlemcen
Département d'informatique
1ère année MI

https://sites.google.com/site/informatiquemessabihi/

Introduction

- Jusqu'à présent, les données utilisées dans nos programmes sont :
  1. incluses dans le programme lui-même, par le programmeur,
  2. entrées à l'exécution par l'utilisateur.
- Mais évidemment, cela ne suffit pas à combler les besoins réels.
  - comment peut-on sauvegarder, dans ce cas-là, les noms et les notes des étudiants, les meilleurs scores des joueurs, les documents de texte qu'on rédige ...
  - nécessité d'un moyen de stockage permanent
- Les fichiers sont là pour combler ce manque. Ils servent à stocker des données de manière permanente, entre deux exécutions d'un programme.

Introduction

- Jusqu'à présent, les données utilisées dans nos programmes sont :
  1. incluses dans le programme lui-même, par le programmeur,
  2. entrées à l'exécution par l'utilisateur.
- Mais évidemment, cela ne suffit pas à combler les besoins réels.
  - comment peut-on sauvegarder, dans ce cas-là, les noms et les notes des étudiants, les meilleurs scores des joueurs, les documents textes qu'on rédige...
  - nécessité d'un moyen de stockage permanent
- Les fichiers sont là pour combler ce manque. Ils servent à stocker des données de manière permanente, entre deux exécutions d'un programme.

Introduction

- Jusqu'à présent, les données utilisées dans nos programmes sont :
  1. incluses dans le programme lui-même, par le programmeur,
  2. entrées à l'exécution par l'utilisateur.
- Mais évidemment, cela ne suffit pas à combler les besoins réels.
  - comment peut-on sauvegarder, dans ce cas-là, les noms et les notes des étudiants, les meilleurs scores des joueurs, les documents textes qu'on rédige...
  - nécessité d'un moyen de stockage permanent
- Les fichiers sont là pour combler ce manque. Ils servent à stocker des données de manière permanente, entre deux exécutions d'un programme.

Introduction

- Jusqu'à présent, les données utilisées dans nos programmes sont :
  1. incluses dans le programme lui-même, par le programmeur,
  2. entrées à l'exécution par l'utilisateur.
- Mais évidemment, cela ne suffit pas à combler les besoins réels.
  - comment peut-on sauvegarder, dans ce cas-là, les noms et les notes des étudiants, les meilleurs scores des joueurs, les documents textes qu'on rédige...
  - nécessité d'un moyen de stockage permanent
- Les fichiers sont là pour combler ce manque. Ils servent à stocker des données de manière permanente, entre deux exécutions d'un programme.

Introduction

- Jusqu'à présent, les données utilisées dans nos programmes sont :
  1. incluses dans le programme lui-même, par le programmeur,
  2. entrées à l'exécution par l'utilisateur.
- Mais évidemment, cela ne suffit pas à combler les besoins réels.
  - comment peut-on sauvegarder, dans ce cas-là, les noms et les notes des étudiants, les meilleurs scores des joueurs, les documents textes qu'on rédige...
  - nécessité d'un moyen de stockage permanent
- Les fichiers sont là pour combler ce manque. Ils servent à stocker des données de manière permanente, entre deux exécutions d'un programme.

Introduction

- Jusqu'à présent, les données utilisées dans nos programmes sont :
  1. incluses dans le programme lui-même, par le programmeur,
  2. entrées à l'exécution par l'utilisateur.
- Mais évidemment, cela ne suffit pas à combler les besoins réels.
  - comment peut-on sauvegarder, dans ce cas-là, les noms et les notes des étudiants, les meilleurs scores des joueurs, les documents textes qu'on rédige...
  - nécessité d'un moyen de stockage permanent
- Les fichiers sont là pour combler ce manque. Ils servent à stocker des données de manière permanente, entre deux exécutions d'un programme.

Introduction

- Jusqu'à présent, les données utilisées dans nos programmes sont :
  1. incluses dans le programme lui-même, par le programmeur,
  2. entrées à l'exécution par l'utilisateur.
- Mais évidemment, cela ne suffit pas à combler les besoins réels.
  - comment peut-on sauvegarder, dans ce cas-là, les noms et les notes des étudiants, les meilleurs scores des joueurs, les documents textes qu'on rédige...
  - nécessité d'un moyen de stockage permanent
- Les fichiers sont là pour combler ce manque. Ils servent à stocker des données de manière permanente, entre deux exécutions d'un programme.

Les fichiers

- Toute donnée en mémoire externe est organisée sous forme de fichier(s).
- Un fichier est :
  - une séquence d'octets,
  - repéré par un nom (dit nom externe), par exemple "montexte.txt" ou "TP2.c",...
  - enregistré sur un support physique non volatile de l'ordinateur : disque, clé USB, carte sd,...
- Un fichier n'est pas détruit à l'arrêt de l'ordinateur.
- La taille d'un fichier n'est pas précisée à sa création

Les fichiers

- Toute donnée en mémoire externe est organisée sous forme de fichier(s).
- Un fichier est :
  - une séquence d'octets,
  - repéré par un nom (dit nom externe), par exemple "montexte.txt" ou "TP2.c",...
  - enregistré sur un support physique non volatile de l'ordinateur : disque, clé USB, carte sd,...
- Un fichier n'est pas détruit à l'arrêt de l'ordinateur.
- La taille d'un fichier n'est pas précisée à sa création

Les fichiers

- Toute donnée en mémoire externe est organisée sous forme de fichier(s).
- Un fichier est :
  - une séquence d'octets,
  - repéré par un nom (dit nom externe), par exemple "montexte.txt" ou "TP2.c",...
  - enregistré sur un support physique non volatile de l'ordinateur : disque, clé USB, carte sd,...
- Un fichier n'est pas détruit à l'arrêt de l'ordinateur.
- La taille d'un fichier n'est pas précisée à sa création

Les fichiers

- Toute donnée en mémoire externe est organisée sous forme de fichier(s).
- Un fichier est :
  - une séquence d'octets,
  - repéré par un nom (dit nom externe), par exemple "montexte.txt" ou "TP2.c",...
  - enregistré sur un support physique non volatile de l'ordinateur : disque, clé USB, carte sd,...
- Un fichier n'est pas détruit à l'arrêt de l'ordinateur.
- La taille d'un fichier n'est pas précisée à sa création

Les fichiers

- Toute donnée en mémoire externe est organisée sous forme de fichier(s).
- Un fichier est :
  - une séquence d'octets,
  - repéré par un nom (dit nom externe), par exemple "montexte.txt" ou "TP2.c",...
  - enregistré sur un support physique non volatile de l'ordinateur : disque, clé USB, carte sd,...
- Un fichier n'est pas détruit à l'arrêt de l'ordinateur.
- La taille d'un fichier n'est pas précisée à sa création

Les fichiers

- Toute donnée en mémoire externe est organisée sous forme de fichier(s).
- Un fichier est :
  - une séquence d'octets,
  - repéré par un nom (dit nom externe), par exemple "montexte.txt" ou "TP2.c",...
  - enregistré sur un support physique non volatile de l'ordinateur : disque, clé USB, carte sd,...
- Un fichier n'est pas détruit à l'arrêt de l'ordinateur.
- La taille d'un fichier n'est pas précisée à sa création

Déclarer un Fichier

Un ficher est déclaré en C comme suit :

FILE* <nom_interne> ;

où nom_interne

- désigne un identificateur
- est une variable "pointeur" associée à un fichier de nom nom_externe
- permet de réaliser toutes les opérations d'un programme C sur ce fichier.

```
void main()
{
        FILE* f = NULL;
        // manipulation du fichier f
        // ...
}
```

Un lien doit toujours être établi entre ce fichier logique (nom_enterne f) et un fichier physique réel (nom externe "montexte.txt") se trouvant sur un support externe.

Déclarer un Fichier

Un ficher est déclaré en C comme suit :

FILE* <nom_interne> ;

où nom_interne

- désigne un identificateur
- est une variable "pointeur" associée à un fichier de nom nom_externe
- permet de réaliser toutes les opérations d'un programme C sur ce fichier.

```
void main()
{
        FILE* f = NULL;
        // manipulation du fichier f
        // ...
}
```

Un lien doit toujours être établi entre ce fichier logique (nom_enterne f) et un fichier physique réel (nom externe "montexte.txt") se trouvant sur un support externe.

Manipuler un fichier

Les principales opérations permettant de manipuler un fichier en C sont les suivantes :

1. **fopen() :** ouvrir un fichier.
2. **fclose() :** fermer un fichier.
3. **fwrite() :** écrire des données dans un fichier.
4. **fread() :** lire des données à partir d'un fichier.
5. **fseek() :** se positionner à un endroit précis du fichier.

☞Attention

Assurez-vous, dans la suite, que vous incluez bien au moins les bibliothèques **stdio.h** et **stdlib.h** en haut de votre fichier .c

Manipuler un fichier

Les principales opérations permettant de manipuler un fichier en C sont les suivantes :

1. **fopen() :** ouvrir un fichier.
2. **fclose() :** fermer un fichier.
3. **fwrite() :** écrire des données dans un fichier.
4. **fread() :** lire des données à partir d'un fichier.
5. **fseek() :** se positionner à un endroit précis du fichier.

☞**Attention**

Assurez-vous, dans la suite, que vous incluez bien au moins les bibliothèques **stdio.h** et **stdlib.h** en haut de votre fichier .c

Manipuler un fichier

Les principales opérations permettant de manipuler un fichier en C sont les suivantes :

1. **fopen() :** ouvrir un fichier.
2. **fclose() :** fermer un fichier.
3. **fwrite() :** écrire des données dans un fichier.
4. **fread() :** lire des données à partir d'un fichier.
5. **fseek() :** se positionner à un endroit précis du fichier.

☞**Attention**

Assurez-vous, dans la suite, que vous incluez bien au moins les bibliothèques **stdio.h** et **stdlib.h** en haut de votre fichier .c

Manipuler un fichier

Les principales opérations permettant de manipuler un fichier en C sont les suivantes :

1. **fopen() :** ouvrir un fichier.
2. **fclose() :** fermer un fichier.
3. **fwrite() :** écrire des données dans un fichier.
4. **fread() :** lire des données à partir d'un fichier.
5. **fseek() :** se positionner à un endroit précis du fichier.

☞Attention

Assurez-vous, dans la suite, que vous incluez bien au moins les bibliothèques **stdio.h** et **stdlib.h** en haut de votre fichier .c

Manipuler un fichier

Les principales opérations permettant de manipuler un fichier en C sont les suivantes :

1. **fopen() :** ouvrir un fichier.
2. **fclose() :** fermer un fichier.
3. **fwrite() :** écrire des données dans un fichier.
4. **fread() :** lire des données à partir d'un fichier.
5. **fseek() :** se positionner à un endroit précis du fichier.

☞**Attention**

Assurez-vous, dans la suite, que vous incluez bien au moins les bibliothèques **stdio.h** et **stdlib.h** en haut de votre fichier .c

Manipuler un fichier

Les principales opérations permettant de manipuler un fichier en C sont les suivantes :

1. **fopen() :** ouvrir un fichier.
2. **fclose() :** fermer un fichier.
3. **fwrite() :** écrire des données dans un fichier.
4. **fread() :** lire des données à partir d'un fichier.
5. **fseek() :** se positionner à un endroit précis du fichier.

☞Attention

Assurez-vous, dans la suite, que vous incluez bien au moins les bibliothèques **stdio.h** et **stdlib.h** en haut de votre fichier .c

Manipuler un fichier

Les principales opérations permettant de manipuler un fichier en C sont les suivantes :

1. **fopen() :** ouvrir un fichier.
2. **fclose() :** fermer un fichier.
3. **fwrite() :** écrire des données dans un fichier.
4. **fread() :** lire des données à partir d'un fichier.
5. **fseek() :** se positionner à un endroit précis du fichier.

☞**Attention**
Assurez-vous, dans la suite, que vous incluez bien au moins les bibliothèques **stdio.h** et **stdlib.h** en haut de votre fichier .c

Ouvrir un fichier : fopen()

L'ouverture d'un fichier se fait à l'aide de la fonction **fopen** de prototype :

```
FILE *fopen(char *nom_externe, char *mode);
```

- **nom_externe** est une chaîne de caractères contenant le nom du fichier à ouvrir.
- **mode** est une chaîne de caractères définissant le type et le mode d'accès du fichier. C'est-à-dire une indication qui mentionne ce que vous voulez faire : seulement écrire dans le fichier, seulement le lire, ou les deux à la fois.
- **fopen** retourne le **nom_interne** du fichier en cas de succès et **NULL** dans le cas contraire.

Ouvrir un fichier : fopen()

L'ouverture d'un fichier se fait à l'aide de la fonction **fopen** de prototype :

```
FILE *fopen(char *nom_externe, char *mode);
```

- **nom_externe** est une chaîne de caractères contenant le nom du fichier à ouvrir.
- **mode** est une chaîne de caractères définissant le type et le mode d'accès du fichier. C'est-à-dire une indication qui mentionne ce que vous voulez faire : seulement écrire dans le fichier, seulement le lire, ou les deux à la fois.
- **fopen** retourne le **nom_interne** du fichier en cas de succès et **NULL** dans le cas contraire.

Ouvrir un fichier : fopen()

L'ouverture d'un fichier se fait à l'aide de la fonction **fopen** de prototype :

```
FILE *fopen(char *nom_externe, char *mode);
```

- **nom_externe** est une chaîne de caractères contenant le nom du fichier à ouvrir.
- **mode** est une chaîne de caractères définissant le type et le mode d'accès du fichier. C'est-à-dire une indication qui mentionne ce que vous voulez faire : seulement écrire dans le fichier, seulement le lire, ou les deux à la fois.
- **fopen** retourne le **nom_interne** du fichier en cas de succès et **NULL** dans le cas contraire.

Ouvrir un fichier : fopen()

L'ouverture d'un fichier se fait à l'aide de la fonction **fopen** de prototype :

```
FILE *fopen(char *nom_externe, char *mode);
```

- **nom_externe** est une chaîne de caractères contenant le nom du fichier à ouvrir.
- **mode** est une chaîne de caractères définissant le type et le mode d'accès du fichier. C'est-à-dire une indication qui mentionne ce que vous voulez faire : seulement écrire dans le fichier, seulement le lire, ou les deux à la fois.
- **fopen** retourne le **nom_interne** du fichier en cas de succès et **NULL** dans le cas contraire.

Ouvrir un fichier : modes d'accès

```
FILE* f1 = NULL, f2 = NULL, f3 = NULL;
f1 = fopen("montexte1.txt", "r");          // lecture seule
f2 = fopen("montexte2.txt", "w");          // ecriture seule
f3 = fopen("montexte3.txt", "a");          // mode d'ajout
```

- **"r" : lecture seule**. Pour lire le contenu du fichier, mais pas y écrire. Le fichier doit avoir été créé au préalable. Le curseur associé au fichier est positionné au début du fichier.
- **"w" : écriture seule**. Pour écrire dans le fichier, mais pas lire son contenu. Si le fichier existe il sera détruit. S'il n'existe pas, il sera créé. Le curseur est positionné au début du fichier.
- **"a" : mode d'ajout**. Pour écrire à la fin du fichier quelle que soit la position courante du curseur. Si le fichier n'existe pas, il sera créé.

Ouvrir un fichier : modes d'accès

```
FILE* f1 = NULL, f2 = NULL, f3 = NULL;
f1 = fopen("montexte1.txt", "r");          // lecture seule
f2 = fopen("montexte2.txt", "w");          // ecriture seule
f3 = fopen("montexte3.txt", "a");          // mode d'ajout
```

- **"r" : lecture seule**. Pour lire le contenu du fichier, mais pas y écrire. Le fichier doit avoir été créé au préalable. Le curseur associé au fichier est positionné au début du fichier.

- **"w" : écriture seule**. Pour écrire dans le fichier, mais pas lire son contenu. Si le fichier existe il sera détruit. S'il n'existe pas, il sera créé. Le curseur est positionné au début du fichier.

- **"a" : mode d'ajout**. Pour écrire à la fin du fichier quelle que soit la position courante du curseur. Si le fichier n'existe pas, il sera créé.

Ouvrir un fichier : modes d'accès

```
FILE* f1 = NULL, f2 = NULL, f3 = NULL;
f1 = fopen("montexte1.txt", "r");          // lecture seule
f2 = fopen("montexte2.txt", "w");          // ecriture seule
f3 = fopen("montexte3.txt", "a");          // mode d'ajout
```

- **"r" : lecture seule**. Pour lire le contenu du fichier, mais pas y écrire. Le fichier doit avoir été créé au préalable. Le curseur associé au fichier est positionné au début du fichier.

- **"w" : écriture seule**. Pour écrire dans le fichier, mais pas lire son contenu. Si le fichier existe il sera détruit. S'il n'existe pas, il sera créé. Le curseur est positionné au début du fichier.

- **"a" : mode d'ajout**. Pour écrire à la fin du fichier quelle que soit la position courante du curseur. Si le fichier n'existe pas, il sera créé.

Ouvrir un fichier : modes d'accès

```
FILE* f1 = NULL, f2 = NULL, f3 = NULL;
f1 = fopen("montexte1.txt", "r+");// lecture et ecriture
f2 = fopen("montexte2.txt", "w+");// lecture et ecriture,
    avec suppression du contenu au prealable
f3 = fopen("montexte3.txt", "a+");// ajout en lecture/
    ecriture a la fin
```

- **"r+" : lecture et écriture**. Identique au mode **"r"** avec possibilité d'écriture
- **"w+" : lecture et écriture, avec suppression du contenu au préalable**. Identique au mode **"w"** avec possibilité de lecture.
  ☞Attention ! Le fichier est d'abord vidé de son contenu.
- **"a+" : ajout en lecture/écriture à la fin**. Identique au mode **"a"** avec possibilité de lecture.

Ouvrir un fichier : modes d'accès

```
FILE* f1 = NULL, f2 = NULL, f3 = NULL;
f1 = fopen("montexte1.txt", "r+");// lecture et ecriture
f2 = fopen("montexte2.txt", "w+");// lecture et ecriture,
    avec suppression du contenu au prealable
f3 = fopen("montexte3.txt", "a+");// ajout en lecture/
    ecriture a la fin
```

- **"r+" : lecture et écriture**. Identique au mode **"r"** avec possibilité d'écriture
- **"w+" : lecture et écriture, avec suppression du contenu au préalable**. Identique au mode **"w"** avec possibilité de lecture.
  ☞Attention ! Le fichier est d'abord vidé de son contenu.
- **"a+" : ajout en lecture/écriture à la fin**. Identique au mode **"a"** avec possibilité de lecture.

Ouvrir un fichier : modes d'accès

```
FILE* f1 = NULL, f2 = NULL, f3 = NULL;
f1 = fopen("montexte1.txt", "r+");// lecture et ecriture
f2 = fopen("montexte2.txt", "w+");// lecture et ecriture,
    avec suppression du contenu au prealable
f3 = fopen("montexte3.txt", "a+");// ajout en lecture/
    ecriture a la fin
```

- **"r+" : lecture et écriture**. Identique au mode **"r"** avec possibilité d'écriture

- **"w+" : lecture et écriture, avec suppression du contenu au préalable**. Identique au mode **"w"** avec possibilité de lecture.
☞Attention ! Le fichier est d'abord vidé de son contenu.

- **"a+" : ajout en lecture/écriture à la fin**. Identique au mode **"a"** avec possibilité de lecture.

Ouvrir un fichier : exemple d'une bonne pratique

Ouvrir un fichier revient à créer un lien entre le **nom_interne** (f) et le **nom_externe** ("montexte.txt") :

```
void main()
{
    FILE* f = NULL;
    f = fopen("montexte.txt", "r+");
    if (f != NULL)
    {
        // On peut lire et ecrire dans le fichier
    }
    else
    {
      // On affiche un message d'erreur si on veut
      printf("Impossible d'ouvrir le fichier montexte.txt");
    }
}
```

- Si l'ouverture a fonctionné (si le pointeur est différent de NULL), alors on peut s'amuser à lire et écrire dans le fichier.
- Si le pointeur vaut NULL, c'est que l'ouverture du fichier a échoué. On ne peut donc pas continuer (afficher un message d'erreur).

Ouvrir un fichier : exemple d'une bonne pratique

Ouvrir un fichier revient à créer un lien entre le **nom_interne** (f) et le **nom_externe** ("montexte.txt") :

```
void main()
{
    FILE* f = NULL;
    f = fopen("montexte.txt", "r+");
    if (f != NULL)
    {
        // On peut lire et ecrire dans le fichier
    }
    else
    {
      // On affiche un message d'erreur si on veut
      printf("Impossible d'ouvrir le fichier montexte.txt");
    }
}
```

- Si l'ouverture a fonctionné (si le pointeur est différent de NULL), alors on peut s'amuser à lire et écrire dans le fichier.
- Si le pointeur vaut NULL, c'est que l'ouverture du fichier a échoué. On ne peut donc pas continuer (afficher un message d'erreur).

Ouvrir un fichier : chemins des fichiers

```
FILE* f1, f2, f3, f4;
f1 = fopen("montexte1.txt", "r");          // meme dossier
f2 = fopen("TP2/monTP2.c", "w");           // chemin relatif
f3 = fopen("C:\\Mes_TP\\TP3\\source.h", "a");//chemin absolu
f4 = fopen("/home/Toto/Loulou/TD.doc", "a"); // sous linux
```

- **f1** doit être situé dans le même dossier que l'exécutable (.exe)
- **f2** est situé dans un sous-dossier appelé "TP2". c'est ce qu'on on appelle **chemin relatif**. Peu importe l'endroit où est installé votre programme cela fonctionnera toujours.
- Il est aussi possible d'ouvrir un autre fichier n'importe où ailleurs sur le disque dur en utilisant ce qu'on appelle **chemin absolu** (par exemple le fichier **f3**).
- Le défaut des chemins absolus, c'est qu'ils ne fonctionnent que sur un OS précis. Sous Linux par exemple, on aurait dû écrire un chemin à-la-linux, comme pour le fichier **f4**.

Ouvrir un fichier : chemins des fichiers

```
FILE* f1, f2, f3, f4;
f1 = fopen("montexte1.txt", "r");          // meme dossier
f2 = fopen("TP2/monTP2.c", "w");           // chemin relatif
f3 = fopen("C:\\Mes_TP\\TP3\\source.h", "a");//chemin absolu
f4 = fopen("/home/Toto/Loulou/TD.doc", "a"); // sous linux
```

- **f1** doit être situé dans le même dossier que l'exécutable (.exe)
- **f2** est situé dans un sous-dossier appelé "TP2". c'est ce qu'on on appelle **chemin relatif**. Peu importe l'endroit où est installé votre programme cela fonctionnera toujours.
- Il est aussi possible d'ouvrir un autre fichier n'importe où ailleurs sur le disque dur en utilisant ce qu'on appelle **chemin absolu** (par exemple le fichier **f3**).
- Le défaut des chemins absolus, c'est qu'ils ne fonctionnent que sur un OS précis. Sous Linux par exemple, on aurait dû écrire un chemin à-la-linux, comme pour le fichier **f4**.

Ouvrir un fichier : chemins des fichiers

```
FILE* f1, f2, f3, f4;
f1 = fopen("montexte1.txt", "r");          // meme dossier
f2 = fopen("TP2/monTP2.c", "w");           // chemin relatif
f3 = fopen("C:\\Mes_TP\\TP3\\source.h", "a");//chemin absolu
f4 = fopen("/home/Toto/Loulou/TD.doc", "a"); // sous linux
```

- **f1** doit être situé dans le même dossier que l'exécutable (.exe)
- **f2** est situé dans un sous-dossier appelé "TP2". c'est ce qu'on on appelle **chemin relatif**. Peu importe l'endroit où est installé votre programme cela fonctionnera toujours.
- Il est aussi possible d'ouvrir un autre fichier n'importe où ailleurs sur le disque dur en utilisant ce qu'on appelle **chemin absolu** (par exemple le fichier **f3**).
- Le défaut des chemins absolus, c'est qu'ils ne fonctionnent que sur un OS précis. Sous Linux par exemple, on aurait dû écrire un chemin à-la-linux, comme pour le fichier **f4**.

Ouvrir un fichier : chemins des fichiers

```
FILE* f1, f2, f3, f4;
f1 = fopen("montexte1.txt", "r");          // meme dossier
f2 = fopen("TP2/monTP2.c", "w");           // chemin relatif
f3 = fopen("C:\\Mes_TP\\TP3\\source.h", "a");//chemin absolu
f4 = fopen("/home/Toto/Loulou/TD.doc", "a"); // sous linux
```

- **f1** doit être situé dans le même dossier que l'exécutable (.exe)
- **f2** est situé dans un sous-dossier appelé "TP2". c'est ce qu'on on appelle **chemin relatif**. Peu importe l'endroit où est installé votre programme cela fonctionnera toujours.
- Il est aussi possible d'ouvrir un autre fichier n'importe où ailleurs sur le disque dur en utilisant ce qu'on appelle **chemin absolu** (par exemple le fichier **f3**).
- Le défaut des chemins absolus, c'est qu'ils ne fonctionnent que sur un OS précis. Sous Linux par exemple, on aurait dû écrire un chemin à-la-linux, comme pour le fichier **f4**.

Fermer un fichier : fclose()

La fermeture d'un fichier se fait par la fonction **fclose** de prototype :

int **fclose**(FILE* <nom_interne>)

```
void main()
{
    FILE* f = NULL;
    f = fopen("montexte.txt", "r+");
    if (f != NULL)
    {
        // On peut lire et ecrire dans le fichier
        //...
        fclose(f); // On ferme le fichier qui a ete ouvert
    }
    else
      printf("Impossible d'ouvrir le fichier montexte.txt");
}
```

- **fclose** retourne zéro en cas de succès
- ☞**Attention !** Il faut toujours penser à fermer son fichier une fois que l'on a fini de travailler avec. Cela permet de libérer de la mémoire.

Fermer un fichier : fclose()

La fermeture d'un fichier se fait par la fonction **fclose** de prototype :

int **fclose**(FILE* <nom_interne>)

```
void main()
{
    FILE* f = NULL;
    f = fopen("montexte.txt", "r+");
    if (f != NULL)
    {
        // On peut lire et ecrire dans le fichier
        //...
        fclose(f); // On ferme le fichier qui a ete ouvert
    }
    else
      printf("Impossible d'ouvrir le fichier montexte.txt");
}
```

- **fclose** retourne zéro en cas de succès
- ☞**Attention !** Il faut toujours penser à fermer son fichier une fois que l'on a fini de travailler avec. Cela permet de libérer de la mémoire.

Écrire dans un fichier

Il existe plusieurs fonctions capables d'écrire dans un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fputc :** écrit un caractère dans le fichier (**un seul** caractère à la fois) ;
2. **fputs :** écrit une chaîne dans le fichier ;
3. **fprintf :** écrit une chaîne « formatée » dans le fichier, fonctionnement quasi-identique à printf.
4. **fwrite :** écrit dans le fichier un certain nombre éléments pointés de taille précise.

Écrire dans un fichier

Il existe plusieurs fonctions capables d'écrire dans un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fputc :** écrit un caractère dans le fichier (**un seul** caractère à la fois) ;
2. **fputs :** écrit une chaîne dans le fichier ;
3. **fprintf :** écrit une chaîne « formatée » dans le fichier, fonctionnement quasi-identique à printf.
4. **fwrite :** écrit dans le fichier un certain nombre éléments pointés de taille précise.

Écrire dans un fichier

Il existe plusieurs fonctions capables d'écrire dans un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fputc :** écrit un caractère dans le fichier (**un seul** caractère à la fois) ;
2. **fputs :** écrit une chaîne dans le fichier ;
3. **fprintf :** écrit une chaîne « formatée » dans le fichier, fonctionnement quasi-identique à printf.
4. **fwrite :** écrit dans le fichier un certain nombre éléments pointés de taille précise.

Écrire dans un fichier

Il existe plusieurs fonctions capables d'écrire dans un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fputc :** écrit un caractère dans le fichier (**un seul** caractère à la fois) ;
2. **fputs :** écrit une chaîne dans le fichier ;
3. **fprintf :** écrit une chaîne « formatée » dans le fichier, fonctionnement quasi-identique à printf.
4. **fwrite :** écrit dans le fichier un certain nombre éléments pointés de taille précise.

Écrire dans un fichier

Il existe plusieurs fonctions capables d'écrire dans un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fputc :** écrit un caractère dans le fichier (**un seul** caractère à la fois) ;
2. **fputs :** écrit une chaîne dans le fichier ;
3. **fprintf :** écrit une chaîne « formatée » dans le fichier, fonctionnement quasi-identique à printf.
4. **fwrite :** écrit dans le fichier un certain nombre éléments pointés de taille précise.

Écrire dans un fichier : fputc()

Cette fonction écrit un caractère à la fois dans le fichier :

int **fputc**(int caractere, FILE* pointeurSurFichier);

```
FILE* f = NULL;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
    fputc('A', f); // Ecriture du caractere 'A'
    fclose(f);
}
else
  printf("Impossible d'ouvrir le fichier montexte.txt");
```

Elle prend deux paramètres :

- Le caractère à écrire (de type int).
- Le pointeur sur le fichier dans lequel écrire.

La fonction **fputc** retourne un int, c'est un code d'erreur. Il vaut **EOF** si l'écriture a échoué, sinon il a une autre valeur.

Écrire dans un fichier : fputc()

Cette fonction écrit un caractère à la fois dans le fichier :

int **fputc**(int caractere, FILE* pointeurSurFichier);

```
FILE* f = NULL;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
    fputc('A', f); // Ecriture du caractere 'A'
    fclose(f);
}
else
  printf("Impossible d'ouvrir le fichier montexte.txt");
```

Elle prend deux paramètres :

- Le caractère à écrire (de type int).
- Le pointeur sur le fichier dans lequel écrire.

La fonction **fputc** retourne un int, c'est un code d'erreur. Il vaut **EOF** si l'écriture a échoué, sinon il a une autre valeur.

Écrire dans un fichier : fputc()

Cette fonction écrit un caractère à la fois dans le fichier :

int **fputc**(int caractere, FILE* pointeurSurFichier);

```
FILE* f = NULL;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
    fputc('A', f); // Ecriture du caractere 'A'
    fclose(f);
}
else
  printf("Impossible d'ouvrir le fichier montexte.txt");
```

Elle prend deux paramètres :

- Le caractère à écrire (de type int).
- Le pointeur sur le fichier dans lequel écrire.

La fonction **fputc** retourne un int, c'est un code d'erreur. Il vaut **EOF** si l'écriture a échoué, sinon il a une autre valeur.

Écrire dans un fichier : fputs()

Cette fonction écrit une chaîne de caractères dans le fichier :

char* **fputs**(const char* chaine, FILE* pointeurSurFichier) ;

```
FILE* f = NULL;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
  fputs("Ecriture de Toto\n et Loulou dans le fichier",f);
  fclose(f);
}
else
  printf("Impossible d'ouvrir le fichier montexte.txt");
```

Elle prend deux paramètres :

- **chaine :** la chaîne à écrire.
- **pointeurSurFichier :** comme pour fputc, il s'agit de votre pointeur de type FILE* sur le fichier que vous avez ouvert.

La fonction **fputs** renvoie **EOF** s'il y a eu une erreur.

Écrire dans un fichier : fputs()

Cette fonction écrit une chaîne de caractères dans le fichier :

char* **fputs**(const char* chaine, FILE* pointeurSurFichier);

```
FILE* f = NULL;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
  fputs("Ecriture de Toto\n et Loulou dans le fichier",f);
  fclose(f);
}
else
  printf("Impossible d'ouvrir le fichier montexte.txt");
```

Elle prend deux paramètres :

- **chaine :** la chaîne à écrire.
- **pointeurSurFichier :** comme pour fputc, il s'agit de votre pointeur de type FILE* sur le fichier que vous avez ouvert.

La fonction **fputs** renvoie **EOF** s'il y a eu une erreur.

Écrire dans un fichier : fputs()

Cette fonction écrit une chaîne de caractères dans le fichier :

char* **fputs**(const char* chaine, FILE* pointeurSurFichier);

```
FILE* f = NULL;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
  fputs("Ecriture de Toto\n et Loulou dans le fichier",f);
  fclose(f);
}
else
  printf("Impossible d'ouvrir le fichier montexte.txt");
```

Elle prend deux paramètres :

- **chaine :** la chaîne à écrire.
- **pointeurSurFichier :** comme pour fputc, il s'agit de votre pointeur de type FILE* sur le fichier que vous avez ouvert.

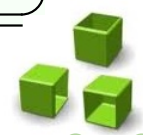
La fonction **fputs** renvoie **EOF** s'il y a eu une erreur.

Écrire dans un fichier : fprintf()

- Elle s'utilise de la même manière que **printf**.
- Il faut juste indiquer un pointeur de **FILE** en premier paramètre.

```
FILE* f = NULL;
int age = 0;
f = fopen("montexte.txt", "r+");
if (f != NULL)
{
     // On demande l'age
     printf("Quel age avez-vous ? ");
     scanf("%d", &age);

     // On l'ecrit dans le fichier
     fprintf(f, "Vous avez %d ans", age);
     fclose(f);
}
else
  printf("Impossible d'ouvrir le fichier montexte.txt");
```



Écrire dans un fichier : fwrite()

Cette fonction écrit une chaîne de caractères dans le fichier :

int **fwrite**(void *p, int taille, int nombre, FILE* nom_interne) ;

```
typedef struct
    { char nom[20];
      int age;
    } Etudiant;
FILE* f = NULL;
Etudiant e1, etu_tab[3];
f = fopen("montexte.txt", "r+");
if (f != NULL)
{ fwrite(&e1, sizeof(Etudiant), 1, f);
      fwrite(etu_tab, sizeof(Etudiant), 3, f);
      fclose(f);
}
```

- **fwrite** écrit dans le fichier "**nombre**" éléments pointés par "**p**", chacun de "**taille**" octets ;
- **fwrite** retourne le nombre d'éléments effectivement écrits ;
- l'ecriture se fait à partir de la position courante du curseur, et déplace celui-ci du nombre d'éléments écrits.

Écrire dans un fichier : fwrite()

Cette fonction écrit une chaîne de caractères dans le fichier :

int **fwrite**(void *p, int taille, int nombre, FILE* nom_interne) ;

```
typedef struct
    { char nom[20];
      int age;
    } Etudiant;
FILE* f = NULL;
Etudiant e1, etu_tab[3];
f = fopen("montexte.txt", "r+");
if (f != NULL)
{ fwrite(&e1, sizeof(Etudiant), 1, f);
      fwrite(etu_tab, sizeof(Etudiant), 3, f);
      fclose(f);
}
```

- **fwrite** écrit dans le fichier "**nombre**" éléments pointés par "**p**", chacun de "**taille**" octets ;
- **fwrite** retourne le nombre d'éléments effectivement écrits ;
- l'ecriture se fait à partir de la position courante du curseur, et déplace celui-ci du nombre d'éléments écrits.

Écrire dans un fichier : fwrite()

Cette fonction écrit une chaîne de caractères dans le fichier :

int **fwrite**(void *p, int taille, int nombre, FILE* nom_interne) ;

```
typedef struct
    { char nom[20];
      int age;
    } Etudiant;
FILE* f = NULL;
Etudiant e1, etu_tab[3];
f = fopen("montexte.txt", "r+");
if (f != NULL)
{ fwrite(&e1, sizeof(Etudiant), 1, f);
      fwrite(etu_tab, sizeof(Etudiant), 3, f);
      fclose(f);
}
```

- **fwrite** écrit dans le fichier "**nombre**" éléments pointés par "**p**", chacun de "**taille**" octets ;
- **fwrite** retourne le nombre d'éléments effectivement écrits ;
- l'ecriture se fait à partir de la position courante du curseur, et déplace celui-ci du nombre d'éléments écrits.

Lire depuis un fichier

Il existe plusieurs fonctions permettant de lire des données à partir d'un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fgetc :** lit un caractère depuis le fichier (**un seul** caractère à la fois) ;
2. **fgets :** lit une chaîne de caractère depuis le fichier ;
3. **fscanf :** lit une chaîne de caractères « formatée » depuis le fichier, fonctionnement quasi-identique à scanf.
4. **fread :** lit depuis le fichier un certain nombre éléments de taille définie, et range les éléments lus en mémoire à une adresse précise.

Lire depuis un fichier

Il existe plusieurs fonctions permettant de lire des données à partir d'un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fgetc :** lit un caractère depuis le fichier (**un seul** caractère à la fois) ;
2. **fgets :** lit une chaîne de caractère depuis le fichier ;
3. **fscanf :** lit une chaîne de caractères « formatée » depuis le fichier, fonctionnement quasi-identique à scanf.
4. **fread :** lit depuis le fichier un certain nombre éléments de taille définie, et range les éléments lus en mémoire à une adresse précise.

Lire depuis un fichier

Il existe plusieurs fonctions permettant de lire des données à partir d'un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fgetc :** lit un caractère depuis le fichier (**un seul** caractère à la fois) ;
2. **fgets :** lit une chaîne de caractère depuis le fichier ;
3. **fscanf :** lit une chaîne de caractères « formatée » depuis le fichier, fonctionnement quasi-identique à scanf.
4. **fread :** lit depuis le fichier un certain nombre éléments de taille définie, et range les éléments lus en mémoire à une adresse précise.

Lire depuis un fichier

Il existe plusieurs fonctions permettant de lire des données à partir d'un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fgetc :** lit un caractère depuis le fichier (**un seul** caractère à la fois) ;
2. **fgets :** lit une chaîne de caractère depuis le fichier ;
3. **fscanf :** lit une chaîne de caractères « formatée » depuis le fichier, fonctionnement quasi-identique à scanf.
4. **fread :** lit depuis le fichier un certain nombre éléments de taille définie, et range les éléments lus en mémoire à une adresse précise.

Lire depuis un fichier

Il existe plusieurs fonctions permettant de lire des données à partir d'un fichier. Il faut choisir celle qui est la plus adaptée à notre cas :

1. **fgetc :** lit un caractère depuis le fichier (**un seul** caractère à la fois) ;
2. **fgets :** lit une chaîne de caractère depuis le fichier ;
3. **fscanf :** lit une chaîne de caractères « formatée » depuis le fichier, fonctionnement quasi-identique à scanf.
4. **fread :** lit depuis le fichier un certain nombre éléments de taille définie, et range les éléments lus en mémoire à une adresse précise.

Écrire dans un fichier : fgetc()

Cette fonction lit un caractère à la fois depuis le fichier :

int **fgetc**(FILE* pointeurSurFichier) ;

```
FILE* f = NULL;
int c = 0;
f = fopen("montexte.txt", "r");
if (f != NULL){
  do{
      c = fgetc(f); // On lit le caractere
      printf("%c", c); // On l'affiche
    } while (c != EOF); // On continue tant que fgetc n'
        a pas retourne EOF (fin de fichier)
}
```

- Cette fonction retourne un int : c'est le caractère qui a été lu.
- Si la fonction n'a pas pu lire de caractère, elle retourne **EOF**.

fgetc avance le curseur d'un caractère à chaque fois que vous en lisez un. Si vous appelez fgetc une seconde fois, la fonction lira donc le second caractère, puis le troisième et ainsi de suite. Vous pouvez donc faire une boucle pour lire les caractères un par un dans le fichier.

Écrire dans un fichier : fgetc()

Cette fonction lit un caractère à la fois depuis le fichier :

int **fgetc**(FILE* pointeurSurFichier) ;

```
FILE* f = NULL;
int c = 0;
f = fopen("montexte.txt", "r");
if (f != NULL){
  do{
      c = fgetc(f); // On lit le caractere
      printf("%c", c); // On l'affiche
    } while (c != EOF); // On continue tant que fgetc n'
      a pas retourne EOF (fin de fichier)
}
```

- Cette fonction retourne un int : c'est le caractère qui a été lu.
- Si la fonction n'a pas pu lire de caractère, elle retourne **EOF**.

fgetc avance le curseur d'un caractère à chaque fois que vous en lisez un. Si vous appelez fgetc une seconde fois, la fonction lira donc le second caractère, puis le troisième et ainsi de suite. Vous pouvez donc faire une boucle pour lire les caractères un par un dans le fichier.

Lire depuis un fichier : fgets()

Cette fonction lit une chaîne depuis le fichier.

char* **fgets**(char* chaine, int nbr_Caracteres, FILE* ptr_Fichier) ;

```
#define TAILLE_MAX 1000
int main()
{ FILE* f = NULL;
  char chaine[TAILLE_MAX] = "";
  f = fopen("montexte.txt", "r");
  if (f != NULL)
        {fgets(chaine, TAILLE_MAX, f); // On lit maximum
            TAILLE_MAX caracteres du fichier, on stocke le
            tout dans "chaine"
    printf("%s", chaine); // On affiche la chaine
    fclose(f);
        }
}
```

- **nbr_Caracteres :** c'est le nombre de caractères à lire. En effet, La fonction fgets s'arrête de lire la ligne si elle contient plus de nbr_Caracteres caractères.
- La fonction lit au maximum une ligne (elle s'arrête au premier \n qu'elle rencontre).

Lire depuis un fichier : fgets()

Cette fonction lit une chaîne depuis le fichier.

char* **fgets**(char* chaine, int nbr_Caracteres, FILE* ptr_Fichier) ;

```
#define TAILLE_MAX 1000
int main()
{ FILE* f = NULL;
  char chaine[TAILLE_MAX] = "";
  f = fopen("montexte.txt", "r");
  if (f != NULL)
        {fgets(chaine, TAILLE_MAX, f); // On lit maximum
            TAILLE_MAX caracteres du fichier, on stocke le
            tout dans "chaine"
    printf("%s", chaine); // On affiche la chaine
    fclose(f);
        }
}
```

- **nbr_Caracteres :** c'est le nombre de caractères à lire. En effet, La fonction fgets s'arrête de lire la ligne si elle contient plus de nbr_Caracteres caractères.
- La fonction lit au maximum une ligne (elle s'arrête au premier \n qu'elle rencontre).

Lire depuis un fichier : fgets()

Cette fonction lit une chaîne depuis le fichier.

char* **fgets**(char* chaine, int nbr_Caracteres, FILE* ptr_Fichier) ;

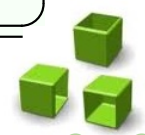
```
#define TAILLE_MAX 1000
int main()
{ FILE* f = NULL;
  char chaine[TAILLE_MAX] = "";
  f = fopen("montexte.txt", "r");
  if (f != NULL)
        {fgets(chaine, TAILLE_MAX, f); // On lit maximum
            TAILLE_MAX caracteres du fichier, on stocke le
            tout dans "chaine"
    printf("%s", chaine); // On affiche la chaine
    fclose(f);
        }
}
```

- **nbr_Caracteres :** c'est le nombre de caractères à lire. En effet, La fonction fgets s'arrête de lire la ligne si elle contient plus de nbr_Caracteres caractères.
- La fonction lit au maximum une ligne (elle s'arrête au premier \n qu'elle rencontre).

Lire depuis un fichier : fgets()

Si vous voulez lire plusieurs lignes, il faudra faire une boucle.

```
#define TAILLE_MAX 1000
int main()
{ FILE* f = NULL;
  char chaine[TAILLE_MAX] = "";
  f = fopen("montexte.txt", "r");
  if (f != NULL)
  {
        while (fgets(chaine, TAILLE_MAX, fichier) != NULL)
            // On lit le fichier tant qu'on ne recoit pas d'
            erreur (NULL)
        {
            printf("%s", chaine); // On affiche la chaene qu
                'on vient de lire
        }
    fclose(f);
  }
}
```

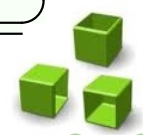


Lire depuis un fichier : fscanf()

- La fonction **fscanf** s'utilise de la même manière que **scanf**.
- Elle lit depuis un fichier qui doit avoir été écrit en respectant un format particulier.

```
int main()
{
    FILE* f = NULL;
    Etudiant e = {"Toto", 19};

    int age = 0;
    f = fopen("montexte.txt", "r");
    if (f != NULL)
    {
        fscanf(fichier, "%s %d", &e.nom, &e.age);
        printf("Nom : %d, Age : %d ans", e.nom, e.age);
        fclose(f);
    }
}
```



Lire depuis un fichier : fread()

Cette fonction permet un certain nombre de données depuis un fichier :

int **fread**(void *p, int taille, int nombre, FILE* nom_interne) ;

```
FILE* f = NULL;
Etudiant e, etu_tab[3];
f = fopen("montexte.txt", "r");
if (f != NULL)
{
    fread(&e, sizeof(Etudiant), 1, f);
    fread(etu_tab, sizeof(Etudiant), 3, f);
    fclose(f);
}
```

- **fread** lit dans le fichier "**nombre**" éléments, chacun de "**taille**" octets, et range les éléments lus en mémoire à l'adresse "**p**".
- **fread** retourne le nombre d'éléments effectivement lus.
- la lecture se fait à partir de la position courante du curseur, et déplace le curseur du nombre d'éléments lus.

Lire depuis un fichier : fread()

Cette fonction permet un certain nombre de données depuis un fichier :

int **fread**(void *p, int taille, int nombre, FILE* nom_interne) ;

```
FILE* f = NULL;
Etudiant e, etu_tab[3];
f = fopen("montexte.txt", "r");
if (f != NULL)
{
    fread(&e, sizeof(Etudiant), 1, f);
    fread(etu_tab, sizeof(Etudiant), 3, f);
    fclose(f);
}
```

- **fread** lit dans le fichier "**nombre**" éléments, chacun de "**taille**" octets, et range les éléments lus en mémoire à l'adresse "**p**".
- **fread** retourne le nombre d'éléments effectivement lus.
- la lecture se fait à partir de la position courante du curseur, et déplace le curseur du nombre d'éléments lus.

Lire depuis un fichier : fread()

Cette fonction permet un certain nombre de données depuis un fichier :

int **fread**(void *p, int taille, int nombre, FILE* nom_interne) ;

```
FILE* f = NULL;
Etudiant e, etu_tab[3];
f = fopen("montexte.txt", "r");
if (f != NULL)
{
    fread(&e, sizeof(Etudiant), 1, f);
    fread(etu_tab, sizeof(Etudiant), 3, f);
    fclose(f);
}
```

- **fread** lit dans le fichier "**nombre**" éléments, chacun de "**taille**" octets, et range les éléments lus en mémoire à l'adresse "**p**".
- **fread** retourne le nombre d'éléments effectivement lus.
- la lecture se fait à partir de la position courante du curseur, et déplace le curseur du nombre d'éléments lus.

Lire depuis un fichier : fread()

Cette fonction permet un certain nombre de données depuis un fichier :

int **fread**(void *p, int taille, int nombre, FILE* nom_interne) ;

```
FILE* f = NULL;
Etudiant e, etu_tab[3];
f = fopen("montexte.txt", "r");
if (f != NULL)
{
    fread(&e, sizeof(Etudiant), 1, f);
    fread(etu_tab, sizeof(Etudiant), 3, f);
    fclose(f);
}
```

- **fread** lit dans le fichier "**nombre**" éléments, chacun de "**taille**" octets, et range les éléments lus en mémoire à l'adresse "**p**".
- **fread** retourne le nombre d'éléments effectivement lus.
- la lecture se fait à partir de la position courante du curseur, et déplace le curseur du nombre d'éléments lus.

Se positionner dans un fichier

Il existe trois fonctions à connaître :

```
long ftell(FILE* ptrFichier);

int fseek(FILE* ptrFichier, long deplacement, int origine);

void rewind(FILE* ptrFichier);
```

- **ftell()** : renvoie la position actuelle du curseur sous la forme d'un long.
- **fseek()** : positionne le curseur à un endroit précis.
- **rewind()** : remet le curseur au début du fichier (c'est équivalent à demander à la fonction fseek de positionner le curseur au début).

Se positionner dans un fichier

Il existe trois fonctions à connaître :

```
long ftell(FILE* ptrFichier);

int fseek(FILE* ptrFichier, long deplacement, int origine);

void rewind(FILE* ptrFichier);
```

- **ftell() :** renvoie la position actuelle du curseur sous la forme d'un long.
- **fseek() :** positionne le curseur à un endroit précis.
- **rewind() :** remet le curseur au début du fichier (c'est équivalent à demander à la fonction fseek de positionner le curseur au début).

Se positionner dans un fichier

Il existe trois fonctions à connaître :

```
long ftell(FILE* ptrFichier);

int fseek(FILE* ptrFichier, long deplacement, int origine);

void rewind(FILE* ptrFichier);
```

- **ftell() :** renvoie la position actuelle du curseur sous la forme d'un long.
- **fseek() :** positionne le curseur à un endroit précis.
- **rewind() :** remet le curseur au début du fichier (c'est équivalent à demander à la fonction fseek de positionner le curseur au début).

Se positionner dans un fichier

Il existe trois fonctions à connaître :

```
long ftell(FILE* ptrFichier);

int fseek(FILE* ptrFichier, long deplacement, int origine);

void rewind(FILE* ptrFichier);
```

- **ftell() :** renvoie la position actuelle du curseur sous la forme d'un long.
- **fseek() :** positionne le curseur à un endroit précis.
- **rewind() :** remet le curseur au début du fichier (c'est équivalent à demander à la fonction fseek de positionner le curseur au début).

Se positionner dans un fichier : fseek()

La fonction **fseek** permet de déplacer le curseur d'un certain nombre de caractères (indiqué par **deplacement**) à partir de la position indiquée par **origine**.

```
int fseek(FILE* ptrFichier, long deplacement, int origine);
```

- Le nombre **deplacement** peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).
- Quant au nombre **origine**, il prend l'une des trois constantes (généralement des **#define**) listées ci-dessous :
  1. **SEEK_SET** : indique le début du fichier ;
  2. **SEEK_CUR** : indique la position actuelle du curseur ;
  3. **SEEK_END** : indique la fin du fichier.

Se positionner dans un fichier : fseek()

La fonction **fseek** permet de déplacer le curseur d'un certain nombre de caractères (indiqué par **deplacement**) à partir de la position indiquée par **origine**.

```
int fseek(FILE* ptrFichier, long deplacement, int origine);
```

- Le nombre **deplacement** peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).
- Quant au nombre **origine**, il prend l'une des trois constantes (généralement des **#define**) listées ci-dessous :
  1. **SEEK_SET** : indique le début du fichier ;
  2. **SEEK_CUR** : indique la position actuelle du curseur ;
  3. **SEEK_END** : indique la fin du fichier.

Se positionner dans un fichier : fseek()

La fonction **fseek** permet de déplacer le curseur d'un certain nombre de caractères (indiqué par **deplacement**) à partir de la position indiquée par **origine**.

```
int fseek(FILE* ptrFichier, long deplacement, int origine);
```

- Le nombre **deplacement** peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).
- Quant au nombre **origine**, il prend l'une des trois constantes (généralement des **#define**) listées ci-dessous :
  1. **SEEK_SET** : indique le début du fichier ;
  2. **SEEK_CUR** : indique la position actuelle du curseur ;
  3. **SEEK_END** : indique la fin du fichier.

Se positionner dans un fichier : fseek()

La fonction **fseek** permet de déplacer le curseur d'un certain nombre de caractères (indiqué par **deplacement**) à partir de la position indiquée par **origine**.

```
int fseek(FILE* ptrFichier, long deplacement, int origine);
```

- Le nombre **deplacement** peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).
- Quant au nombre **origine**, il prend l'une des trois constantes (généralement des **#define**) listées ci-dessous :
  1. **SEEK_SET** : indique le début du fichier ;
  2. **SEEK_CUR** : indique la position actuelle du curseur ;
  3. **SEEK_END** : indique la fin du fichier.

Se positionner dans un fichier : fseek()

La fonction **fseek** permet de déplacer le curseur d'un certain nombre de caractères (indiqué par **deplacement**) à partir de la position indiquée par **origine**.

```
int fseek(FILE* ptrFichier, long deplacement, int origine);
```

- Le nombre **deplacement** peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).
- Quant au nombre **origine**, il prend l'une des trois constantes (généralement des **#define**) listées ci-dessous :
  1. **SEEK_SET** : indique le début du fichier ;
  2. **SEEK_CUR** : indique la position actuelle du curseur ;
  3. **SEEK_END** : indique la fin du fichier.

Se positionner dans un fichier : fseek()

La fonction **fseek** permet de déplacer le curseur d'un certain nombre de caractères (indiqué par **deplacement**) à partir de la position indiquée par **origine**.

```
int fseek(FILE* ptrFichier, long deplacement, int origine);
```

- Le nombre **deplacement** peut être un nombre positif (pour se déplacer en avant), nul (= 0) ou négatif (pour se déplacer en arrière).
- Quant au nombre **origine**, il prend l'une des trois constantes (généralement des **#define**) listées ci-dessous :
  1. **SEEK_SET** : indique le début du fichier ;
  2. **SEEK_CUR** : indique la position actuelle du curseur ;
  3. **SEEK_END** : indique la fin du fichier.

Se positionner dans un fichier : exemples

```
    FILE* f = NULL;
Etudiant e, etu_tab[3];
f = fopen("toto.txt", "r");
if (f != NULL)
{
    fread(&e, sizeof(Etudiant), 1, f);
    fread(etu_tab, sizeof(Etudiant), 3, f);
    fseek(f, 0, SEEK_SET);
    fseek(f, sizeof(Etudiant), SEEK_CUR);
    fseek(f, -2*sizeof(Etudiant), SEEK_END);
    printf("%d   %d\n", ftell(f), sizeof(Etudiant));
    rewind(f);
    printf("%d\n", ftell(f));
    fclose(f);
}
```

Renommer et supprimer un fichier

```
                        // prototypes :
int rename(const char* ancienNom, const char* nouveauNom);

int remove(const char* fichierASupprimer);

                        // utilisation
void main()
{
    rename("test.txt", "test1.txt");
    remove("test1.txt");
}
```

- **rename :** permet de renommer un fichier ;
- **remove :** permet de supprimer un fichier sans demander de confirmation .

Ces fonctions ne nécessitent pas de pointeur de fichier, il suffira simplement d'indiquer le nom du fichier à renommer ou à supprimer.

Renommer et supprimer un fichier

```
                        // prototypes :
int rename(const char* ancienNom, const char* nouveauNom);

int remove(const char* fichierASupprimer);

                        // utilisation
void main()
{
    rename("test.txt", "test1.txt");
    remove("test1.txt");
}
```

- **rename :** permet de renommer un fichier ;
- **remove :** permet de supprimer un fichier sans demander de confirmation .

Ces fonctions ne nécessitent pas de pointeur de fichier, il suffira simplement d'indiquer le nom du fichier à renommer ou à supprimer.

Renommer et supprimer un fichier

```
                        // prototypes :
int rename(const char* ancienNom, const char* nouveauNom);

int remove(const char* fichierASupprimer);

                        // utilisation
void main()
{
    rename("test.txt", "test1.txt");
    remove("test1.txt");
}
```

- **rename :** permet de renommer un fichier ;
- **remove :** permet de supprimer un fichier sans demander de confirmation .

Ces fonctions ne nécessitent pas de pointeur de fichier, il suffira simplement d'indiquer le nom du fichier à renommer ou à supprimer.

Renommer et supprimer un fichier

```
                        // prototypes :
int rename(const char* ancienNom, const char* nouveauNom);

int remove(const char* fichierASupprimer);

                        // utilisation
void main()
{
    rename("test.txt", "test1.txt");
    remove("test1.txt");
}
```

- **rename :** permet de renommer un fichier ;
- **remove :** permet de supprimer un fichier sans demander de confirmation .

Ces fonctions ne nécessitent pas de pointeur de fichier, il suffira simplement d'indiquer le nom du fichier à renommer ou à supprimer.