Programmation en langage C

Plan

1. Rappels (Types de bases, variables, opérateurs, structures de contrôles,..)
2. Les fonctions
3. Les pointeurs
4. L'allocation dynamique
5. Les chaines de caractères
6. Les types composés (structures, énumérations,…)
7. Les fichiers
8. Les préprocesseurs

Les fichiers

Objectif : lire et écrire dans les fichiers en langage C

1. **Ouvrir et fermer un fichier**
2. **Différentes méthodes de lecture / écriture**
3. **Se déplacer dans un fichier**
4. **Renommer et supprimer un fichier**

Ouvrir et fermer un fichier

- Se servir des fonctions situées dans la bibliothèque **stdio**

```
#include <stdlib.h>
#include <stdio.h>
```

Ouvrir et fermer un fichier

Les étapes d'ouverture d'un fichier pour le lire ou y écrire :

1. Appeler de la fonction d'ouverture de fichier **fopen** qui nous renvoie un pointeur sur le fichier.
2. Vérifier si l'ouverture a réussi (c'est-à-dire si le fichier existait) en testant la valeur du pointeur qu'on a reçu. **Si le pointeur vaut NULL** , c'est que l'ouverture du fichier n'a pas fonctionné, dans ce cas on ne peut pas continuer (on affiche un message d'erreur).
3. Si l'ouverture a fonctionné (**si le pointeur est différent de NULL**), alors on peut lire et écrire dans le fichier

4. Fermer le fichier avec la fonction **fclose** .

Ouvrir et fermer un fichier

fopen : ouverture du fichier

- Prototype de la fonction :

```
FILE* fopen(const char* nomDuFichier, const char* modeOuverture);
```

Deux paramètres d'entrée :

- le nom du fichier à ouvrir ;
- le mode d'ouverture du fichier (seulement écrire dans le fichier, seulement le lire, ou les deux à la fois).

Ouvrir et fermer un fichier

fopen : ouverture du fichier

- Prototype de la fonction :

```
FILE* fopen(const char* nomDuFichier, const char* modeOuverture);
```

Le type du renvoie :

un pointeur sur FILE : c'est un pointeur sur une structure de type FILE . Cette structure est définie dans stdio.h .

Il faut récupérer ce pointeur pour pouvoir ensuite lire et écrire dans le fichier.

Ouvrir et fermer un fichier

fopen : ouverture du fichier

- **Création d'un pointeur de FILE**

```
int main()
{
    FILE* fichier = NULL;
    return 0;
}
```

Ouvrir et fermer un fichier

fopen : ouverture du fichier

- Le paramètre **modeOuverture**

"r" : lecture seule. Vous pourrez lire le contenu du fichier, mais pas y écrire. Le fichier doit avoir été créé au préalable.

"w" : écriture seule. Vous pourrez écrire dans le fichier, mais pas lire son contenu. Si le fichier n'existe pas, il sera créé.

"a" : mode d'ajout. Vous écrirez dans le fichier, en partant de la fin du fichier. Vous ajouterez donc du texte à la fin du fichier. Si le fichier n'existe pas, il sera créé.

"r+" : lecture et écriture. Vous pourrez lire et écrire dans le fichier. Le fichier doit avoir été créé au préalable.

Ouvrir et fermer un fichier

fopen : ouverture du fichier

- Le paramètre **modeOuverture**

"w+" : lecture et écriture, avec suppression du contenu au préalable. Le fichier est donc d'abord vidé de son contenu, vous pouvez y écrire, et le lire ensuite. Si le fichier n'existe pas, il sera créé.

"a+" : ajout en lecture / écriture à la fin. Vous écrivez et lisez du texte à partir de la fin du fichier. Si le fichier n'existe pas, il sera créé.

Plusieurs autres modes d'ouverture :"rb" , "wb" , "ab" , "rb+" , "wb+" , "ab+",….

Ouvrir et fermer un fichier

fopen : ouverture du fichier

- **Exemple :**

  Ouverture du fichier test.txt en mode "r+" (lecture / écriture)

```
int main()
{
    FILE* fichier = NULL;
    fichier = fopen("test.txt", "r+");
return 0;
}
```

Le pointeur fichier devient un pointeur sur test.txt

Ouvrir et fermer un fichier

fopen : ouverture du fichier

Remarques :

1. Le fichier ne doit pas nécessairement avoir l'extension « .txt »

2. Le fichier doit être dans le même répertoire que l'exécutable, sinon il faut définir son chemin (relatif ou absolu) :

```
fichier = fopen("dossier/test.txt", "r+");

fichier = fopen("C:\\Program Files\\Notepad++\\readme.txt", "r+");
```

Ouvrir et fermer un fichier

fopen : ouverture du fichier

➢ Tester l'ouverture du fichier

- Le pointeur fichier doit contenir l'adresse de la structure de type FILE qui sert de descripteur de fichier. Celui-ci est chargé en mémoire par la fonction fopen() .

- Juste après l'ouverture du fichier, il faut impérativement vérifier si l'ouverture a réussi ou non.

- **Deux possibilités :**
  - **soit l'ouverture a réussi, si le pointeur vaut NULL ;**
  - **soit l'ouverture a échoué si le pointeur vaut autre chose que NULL**

Ouvrir et fermer un fichier

fopen : ouverture du fichier

- **Tester l'ouverture du fichier**

```
int main()
{
	FILE* fichier = NULL;
	fichier = fopen("test.txt", "r+");
	if (fichier != NULL)
	{
		// On peut lire et écrire dans le fichier
	}
	else
	{
		// On affiche un message d'erreur
		printf("Impossible d'ouvrir le fichier test.txt");
	}
	return 0;
}
```

Ouvrir et fermer un fichier

fclose : fermeture du fichier

Permet de libérer la mémoire, c'est-à-dire supprimer votre fichier chargé dans la mémoire vive.

- **Prototype de la fonction :**

```
int fclose(FILE* pointeurSurFichier);
```

- **1 paramètre d'entrée :** le pointeur sur le fichier.
- **Renvoie un int** qui indique si la fonction a réussi à fermer le fichier. Cet int vaut :
  - **0 : si la fermeture a réussi ;**
  - **EOF : si la fermeture a échoué**. EOF est un define situé dans stdio.h qui correspond à un nombre spécial, utilisé pour dire soit qu'il y a eu une erreur, soit que nous sommes arrivés à la fin du fichier. Dans le cas présent cela signifie qu'il y a eu une erreur.

Ouvrir et fermer un fichier

Ouverture et fermeture d'un fichier

```
int main()
{
	FILE* fichier = NULL;
	fichier = fopen("test.txt", "r+");
	if (fichier != NULL)
	{
		// On peut lire et écrire dans le fichier
		fclose(fichier); // On ferme le fichier qui a été ouvert
	}
	else
	{
		// On affiche un message d'erreur
		printf("Impossible d'ouvrir le fichier test.txt");
	}
	return 0;
}
```

Les fichiers

Objectif : lire et écrire dans les fichiers en langage C

1. **Ouvrir et fermer un fichier**
2. **Différentes méthodes de lecture / écriture**
3. **Se déplacer dans un fichier**
4. **Renommer et supprimer un fichier**

Méthodes de lecture / écriture

Ecrire dans le fichier

Il existe plusieurs fonctions capables d'écrire dans un fichier. Nous allons étudier ces trois fonctions :

fputc : écrit un caractère dans le fichier (**UN SEUL caractère à la fois**) ;

fputs : écrit une chaîne dans le fichier ;

fprintf : écrit une chaîne « formatée » dans le fichier, fonctionnement **quasi-identique à printf** .

Méthodes de lecture / écriture

Ecrire dans le fichier

- **Prototype de la fonction fputc :**

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

Deux paramètres d'entrée :

1. **caractere :** le caractère à écrire
2. **pointeurSurFichier :** le pointeur sur le fichier dans lequel il faut écrire.

Méthodes de lecture / écriture

Ecrire dans le fichier

- **Prototype de la fonction fputc :**

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

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

Méthodes de lecture / écriture

Ecrire dans le fichier

- **Exemple avec fputc :** écriture de la lettre 'A' dans test.txt (si le fichier existe, il est remplacé ; s'il n'existe pas, il est créé).

```
int main()
{
    FILE* fichier = NULL;
    fichier = fopen("test.txt", "w");
    if (fichier != NULL)
    {
        fputc('A', fichier); // Écriture du caractère A
        fclose(fichier);
    }
    return 0;
}
```

Méthodes de lecture / écriture

Ecrire dans le fichier

- **Prototype de la fonction fputs :**

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

Deux paramètres d'entrée :

1. **chaine :** la chaîne à écrire, elle est considérée comme une constante, c.à.d. la fonction ne peut pas modifier le contenu de la chaîne.
2. **pointeurSurFichier :** le pointeur sur le fichier dans lequel il faut écrire.

Méthodes de lecture / écriture

Ecrire dans le fichier

- **Exemple avec fputs :** écriture d'un texte dans test.txt

```
int main()
{
    FILE* fichier = NULL;
    fichier = fopen("test.txt", "w");
    if (fichier != NULL)
    {
        fputs("Bonour tous le monde", fichier);
        fclose(fichier);
    }
    return 0;
}
```

Méthodes de lecture / écriture

Ecrire dans le fichier

- **La fonction fprintf :**

S'utilise de la même manière que printf, excepté le fait qu'il faut indiquer un pointeur de FILE en premier paramètre.

Méthodes de lecture / écriture

Ecrire dans le fichier

- **Exemple avec fprintf :**

```
int main()
{
	FILE* fichier = NULL;
	int age = 0;
	fichier = fopen("test.txt", "w");
	if (fichier != NULL)
	{
		 // On demande l'âge
		printf("Quel age avez-vous ? ");
		scanf("%d", &age);
		// On l'écrit dans le fichier
		fprintf(fichier, « Vous avez %d ans", age);
		fclose(fichier);
	}
	return 0;
}
```

Méthodes de lecture / écriture

Lire un fichier

Il existe plusieurs fonctions capables de lire un fichier. Nous allons étudier ces trois fonctions :

fgetc : lit un caractère ;

fgets : lit une chaîne ;

fscanf : lit une chaîne formatée.

Méthodes de lecture / écriture

Lire un fichier : la fonction fgetc

- **Prototype de la fonction fgetc :**

```
int fgetc(FILE* pointeurDeFichier);
```

Un paramètre d'entrée : pointeurSurFichier, le pointeur sur le fichier qu'on va lire.

Type du renvoie : la 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 .

Méthodes de lecture / écriture

Lire un fichier

Comment savoir quel caractère on lit ?

- **Un « curseur »** qui avance au fur et à mesure de la lecture du fichier
- **On peut savoir à quelle position le curseur est situé** dans le fichier et également comment modifier la position du curseur (pour le remettre au début du fichier par exemple, ou le placer à un caractère précis, comme le dixième caractère).
- **fgetc avance le curseur d'un seul 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.

Méthodes de lecture / écriture

Lire un fichier

Exemple avec fgetc : lecture de tous les caractères d'un fichier un à un et les afficher à chaque fois à l'écran. La boucle s'arrête quand fgetc renvoie EOF (« End Of File » c.à.d. « fin du fichier »).

Méthodes de lecture / écriture

Lire un fichier

```
int main()
{
	FILE* fichier = NULL;
	int caractereActuel = 0;
	fichier = fopen("test.txt", "r");
	if (fichier != NULL)
	{
		do // Boucle de lecture des caractères un à un
		{
			caractereActuel = fgetc(fichier); // On lit le caractère
			printf("%c", caractereActuel); // On l'affiche
		} while (caractereActuel != EOF); // On continue tant que fgetc n'a pas retourné
                                                  EOF (fin de fichier)
		fclose(fichier);
	}
	return 0;
}
```

Méthodes de lecture / écriture

Lire un fichier : la fonction fgets

Lit une chaîne dans le fichier. Elle lit au maximum une ligne (elle s'arrête au premier \n qu'elle rencontre). Pour lire plusieurs lignes, il faudra faire une boucle.

➢ **Prototype de la fonction fgets :**

```
char* fgets(char* chaine, int nbreDeCaracteresALire, FILE* pointeurSurFichier);
```

Trois paramètres d'entrée :

nbreDeCaracteresALire : le nombre de caractères à lire. Permet de demander demande à la fonction de s'arrêter de lire la ligne si elle contient plus de X caractères.
chaine : la chaine ou on va stocker le contenu qu'on va lire
pointeurSurFichier, le pointeur sur le fichier qu'on va lire.

Méthodes de lecture / écriture

Lire un fichier

Exemple (1) avec fgets : lecture d’une seule ligne. On crée une chaîne suffisamment grande pour stocker le contenu de la ligne qu'on va lire.

Méthodes de lecture / écriture

Lire un fichier

```
#define TAILLE_MAX 1000 // Tableau de taille 1000
int main()
{
    FILE* fichier = NULL;
    char chaine[TAILLE_MAX] = ""; // Chaîne vide de taille TAILLE_MAX
    fichier = fopen("test.txt", "r");
    if (fichier != NULL)
    {
        // On lit maximum TAILLE_MAX caractères du fichier, on stocke le tout
           dans "chaine"
        fgets(chaine, TAILLE_MAX, fichier);
        printf("%s", chaine); // On affiche la chaîne
        fclose(fichier);
    }
    return 0;
}
```

Méthodes de lecture / écriture

Lire un fichier

Exemple (1) avec fgets : lecture d'une seule ligne. On crée une chaîne suffisamment grande pour stocker le contenu de la ligne qu'on va lire.

Même résultat avec fgetc, la différence, c'est qu'ici on ne fait pas de boucle. On affiche toute la chaîne d'un coup.

Méthodes de lecture / écriture

Lire un fichier

Exemple (2) avec fgets : lecture de tous le contenu du fichier. On utilisant la boucle while

- La fonction fgets renvoie NULL si la lecture a échouée
- La boucle doit donc s'arrêter dès que fgets se met à renvoyer NULL .

Méthodes de lecture / écriture

Lire un fichier

```
#define TAILLE_MAX 1000 // Tableau de taille 1000
int main()
{
	FILE* fichier = NULL;
	char chaine[TAILLE_MAX] = ""; // Chaîne vide de taille TAILLE_MAX
	fichier = fopen("test.txt", "r");
	if (fichier != NULL)
	{
		// On lit le fichier tant qu'on ne reçoit pas d'erreur (NULL)
		while (fgets(chaine, TAILLE_MAX, fichier) != NULL)
		{
			printf("%s", chaine); // On affiche la chaîne qu'on vient de lire
		}
		fclose(fichier);
	}
	return 0;
```

Méthodes de lecture / écriture

Lire un fichier : la fonction fscanf

Cette fonction a le même principe que la fonction scanf .
Elle lit dans un fichier qui doit avoir été écrit d'une manière précise.

Exemple : lecture d'un fichier qui contient trois nombres séparés par un espace : 15 20 30.

Méthodes de lecture / écriture

Lire un fichier : la fonction fscanf

```
int main()
{
	FILE* fichier = NULL;
	int num[3] = {0}; // Tableau des 3 chiifres
	fichier = fopen("test.txt", "r");
	if (fichier != NULL)
	{
		fscanf(fichier, "%d %d %d", &num[0], &num[1], &num[2]);
		printf("Les numéros sont : %d, %d et %d", num[0], num[1], num[2]);
		fclose(fichier);
	}
	return 0;
}
```

Les numéros sont : 15, 20 et 30

7

Les Fichiers

Les fichiers

Objectif : lire et écrire dans les fichiers en langage C

1. **Ouvrir et fermer un fichier**
2. **Différentes méthodes de lecture / écriture**
3. **Se déplacer dans un fichier**
4. **Renommer et supprimer un fichier**

Se déplacer dans un fichier

- Il existe un curseur qui indique votre position dans le fichier à chaque fois que vous l'ouvrez.

- Le système de curseur permet d'aller lire et écrire à une position précise dans le fichier.

- Trois fonctions à connaître :
  - **ftell :** indique à quelle position vous êtes actuellement dans le fichier ;
  - **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 déplacer dans un fichier

ftell : position dans le fichier

- Cette fonction renvoie la position actuelle du curseur dans le fochier sous la forme d'un long.

- **Prototype de la fonction ftell :**

```
long ftell(FILE* pointeurSurFichier);
```

Un paramètre d'entrée : **pointeurSurFichier,** le pointeur sur le fichier qu'on va lire.

Type du renvoie : la fonction retourne un **long** qui indique la position du curseur dans le fichier.

Se déplacer dans un fichier

fseek: se positionner dans le fichier

➢ **Prototype de la fonction fseek:**

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

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

Le nombre **deplacement** peut être un nombre **positif** (pour se déplacer en avant), **nul** (= 0) ou **négatif** (pour se déplacer en arrière).

Se déplacer dans un fichier

fseek: se positionner dans le fichier

- **Prototype de la fonction fseek:**

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

Le nombre **origine** prend comme valeur l'une des trois constantes (généralement des define) :

SEEK_SET : indique le début du fichier ;

SEEK_CUR : indique la position actuelle du curseur ;

SEEK_END : indique la fin du fichier.

Se déplacer dans un fichier

fseek: se positionner dans le fichier

- **Exemple 1 :** Le code suivant place le curseur deux caractères *après* le début

```
fseek(fichier, 2, SEEK_SET);
```

- **Exemple 2 :** Le code suivant place le curseur quatre caractères *avant* la position courante

```
fseek(fichier, -4, SEEK_CUR);
```

- **Exemple 3 :** Le code suivant place le curseur à la fin du fichier

```
fseek(fichier, 0, SEEK_END);
```

Se déplacer dans un fichier

fseek: se positionner dans le fichier

- Si vous écrivez après avoir fait un fseek qui mène à la fin du fichier, cela ajoutera vos informations à la suite dans le fichier (le fichier sera complété).

- Si vous placez le curseur au début et que vous écrivez, cela écrasera le texte qui se trouvait là.

Se déplacer dans un fichier

rewind: retour au début du fichier

- Cette fonction est équivalente à utiliser fseek pour nous renvoyer à la position 0 dans le fichier.

- **Prototype de la fonction rewind :**

```
void rewind(FILE* pointeurSurFichier);
```

Un paramètre d'entrée : pointeurSurFichier, le pointeur sur le fichier qu'on va lire.

Les fichiers

Objectif : lire et écrire dans les fichiers en langage C

1. **Ouvrir et fermer un fichier**
2. **Différentes méthodes de lecture / écriture**
3. **Se déplacer dans un fichier**
4. **Renommer et supprimer un fichier**

Renommer et supprimer un fichier

Deux fonctions :

- **rename : renomme un fichier.**
- **remove : supprime un fichier.**

- Ces fonctions ne nécessitent pas de pointeur de fichier pour fonctionner. Il suffit simplement d'indiquer le nom du fichier à renommer ou supprimer.

Renommer et supprimer un fichier

rename: renommer un fichier

- **Prototype de la fonction rename :**

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

- La fonction renvoie 0 si elle a réussi à renommer, sinon elle renvoie une valeur différente de 0

```
int main(int argc, char *argv[])
{
    rename("test.txt", "test_renomme.txt");
    return 0;
}
```

Renommer et supprimer un fichier

remove: supprimer un fichier

- **Prototype de la fonction remove :**

```
int remove(const char* fichierASupprimer);
```

- Cette fonction supprime le fichier indiqué sans demander de confirmation. Le fichier n'est pas mis dans la corbeille, il est littéralement supprimé du disque dur.

```
int main(int argc, char *argv[])
{
    remove("test.txt");
    return 0;
}
```