Lire et écrire dans un fichier en langage C

- 30-08-2019
- ESSADDOUKI
- Langage C,
- 4364

Sommaire

Un fichier représente une séquence d'octets, qu'il s'agisse d'un fichier texte ou d'un fichier binaire. Le langage de programmation C permet d'accéder aux fonctions de haut niveau ainsi qu'aux appels de bas niveau (au niveau du système d'exploitation) pour gérer les fichiers sur vos périphériques de stockage. Ce chapitre vous guidera à travers les appels importants pour la gestion de fichiers.

1. fputc et fgetc

fputc

Syntaxe

?
```
1int fputc(int ch, FILE *fp);
```

La fonction **fputc()** est utilisée pour écrire un seul caractère spécifié par le premier argument dans un fichier texte pointé par le pointeur fp. Après avoir écrit un caractère dans le fichier texte, le pointeur de position interne est incrémenté. Si l'écriture réussit, la valeur ASCII du caractère qui a été écrit est renvoyée. En cas d'erreur, il retourne **EOF** i.e -1.

Bien que la définition formelle de **fputc()** dise “il écrit un seul caractère au fichier” ce n'est pas comme ça qu'il est implémenté. En pratique, écrire un seul caractère un par un serait très inefficace et lent. Au lieu de cela, si vous écrivez des caractères un par un dans le fichier, ils sont accumulés dans une mémoire tampon. Une fois le nombre de caractères atteint un nombre raisonnable, ils sont écrits dans le fichier en une seule fois.

Exemple 1 :

```
?
1
2  #include< stdio.h>
3  #include< stdlib.h>
4
5  int main(void)
6  {
7      File *f;
8      f = fopen("test.txt", "w");
9
10     if(f==NULL){
11         printf("Erreur lors de l'ouverture d'un fichier");
12         exit(1);
13     }
14
15     while((ch=getchar())!=EOF){
16         fputc(ch, f);
17     }
18
19     fclose(f);
20
21     return 0;
}
```

Rappelez-vous que la fonction **getchar()** renvoie la valeur **ASCII** du caractère qui vient d'être lu à partir de l'entrée standard et **EOF** lorsqu'un caractère de fin de fichier est rencontré.

fgetc

Syntaxe

```
?
1int fgetc(FILE *fp);
```

Cette fonction est complémentaire à la fonction **fputc()**. Il lit un seul caractère du fichier et incrémente le pointeur de position du fichier. Pour utiliser cette fonction, le fichier doit être ouvert en mode lecture. En cas de succès, il renvoie la valeur ASCII du caractère mais vous pouvez aussi attribuer le résultat à une variable de type char. En cas d'échec ou de fin de fichier, il renvoie **EOF** ou -1.

Tout comme **fputc()** cette fonction utilise la mémoire tampon. Ainsi, au lieu de lire un seul caractère du fichier un par un, un bloc entier de caractères du fichier est lu dans le tampon. Les caractères sont ensuite transmis un par un à la fonction **fgetc()**, jusqu'à ce que le tampon soit vide. S'il reste encore quelques caractères à lire dans le fichier, un bloc de caractères est lu dans le tampon.

Exemple 2 :

```
?
1
2  #include< stdio.h>
3  #include< stdlib.h>
4
5  int main(void)
6  {
7      File *f;
8      f = fopen("test.txt", "r");
9
10     if(f==NULL){
11         printf("Erreur lors de l'ouverture d'un fichier");
12         exit(1);
13     }
14
15     while((ch=fgetc(f))!=EOF){
16         printf("%c", ch);
17     }
18
19     fclose(f);
20
21     return 0;
22 }
```

2. fputs et fgets

fputs

Syntaxe

```
?
1 int fputs(const char *str, FILE *fp);
```

La fonction **fputs()** écrit la chaîne str dans le flux de sortie référencé par **fp**. Elle renvoie une valeur non négative en cas de succès, sinon **EOF** est renvoyé en cas d'erreur.

Exemple 3 :

```
?
1  #include< stdio.h>
2  #include< stdlib.h>
3
4  int main(void)
5  {
6      File *f;
7      char str[20];
8      f = fopen("test.txt", "w");
9
10     if(f==NULL){
11         printf("Erreur lors de l'ouverture d'un fichier");
12         exit(1);
13     }
14
```

```
13      while( gets(str) != NULL){
14          fputs(str, f);
15      }
16
17      fclose(f);
18
19      return 0;
20  }
21
22
```

Voici deux points importants à retenir à propos de la fonction **gets():**

- La fonction **gets()** convertit le caractère de nouvelle ligne entré en caractère nul ('**\0**').
- Quand le caractère de fin de fichier est rencontré, **gets()** renvoie **NULL**.

fgets

Syntaxe

```
1 char *fgets(char *str, int n, FILE *fp);
```

La fonction lit une chaîne du fichier pointé par fp dans la mémoire pointée par **str**.
La fonction lit les caractères du fichier jusqu'à ce qu'une nouvelle ligne ('**\n**') soit lue ou que les caractères **n-1** soient lus ou qu'une fin de fichier soit rencontrée, selon la première éventualité.
Après avoir lue la chaîne de caractères, elle ajoute le caractère null ('**\0**') pour terminer la chaîne de caractères. En cas de succès, elle retourne un pointeur à la **str**. En cas d'erreur ou de fin du fichier, elle renvoie **NULL**.

Exemple 4 :

```
1  #include< stdio.h>
2  #include< stdlib.h>
3
4  int main(void)
5  {
6       File *f;
7       char str[20];
8       f = fopen("test.txt", "r");
9
10      if(f==NULL){
11          printf("Erreur lors de l'ouverture d'un fichier");
12          exit(1);
13      }
14
15      while( fgets(str,20,f) != NULL){
16          puts(str);
17      }
18
19      fclose(f);
```

```
18
19      return 0;
20  }
21
22
```

3. fprintf et fscanf

Jusqu'à présent, nous avons vu comment lire et écrire des caractères et des chaînes dans et à partir du fichier. Dans le monde réel, les données sont composées de nombreux types différents. Dans cette partie, nous verrons comment lire et écrire des données de différents types de manière formatée.

fprintf

Syntaxe

```
1int fprintf(FILE *fp, const char *format [, argument, ...] );
```

La fonction **fprintf()** est identique à **print()**, mais au lieu d'écrire des données sur la console, elle écrit des données formatées dans le fichier. Presque tous les arguments de la fonction **fprintf()** sont identiques à ceux de la fonction **printf()**, à la différence qu'il possède un argument supplémentaire qui est un pointeur de fichier sur le fichier dans lequel la sortie formatée sera écrite.
En cas de succès, le nombre total de caractères écrits dans le fichier est renvoyé. En cas d'erreur, il retourne **EOF**.

Exemple 5 :

```
1  #include< stdio.h>
2  #include< stdlib.h>
3
4  int main(void)
5  {
6      File *f;
7      char nom[20];
8      int age;
9
10     f = fopen("test.txt", "w");
11
12     if(f==NULL){
13         printf("Erreur lors de l'ouverture d'un fichier");
14         exit(1);
15     }
16
17     printf("Saisir votre nom :")
18     scanf("%s",&nom);
19
20     printf("saisir votre age");
21     scanf("%d",&age);
```

```
19
20      fprintf(f, "Nom : %s\tAge : %d\n", nom, age);
21
22      fclose(f);
23
24      return 0;
25  }
26
27
28
```

fscanf

Syntaxe

```
1int fscanf(FILE *fp, const char *format [, argument, ...] );
```

La fonction **fscanf()** est utilisée pour lire une entrée formatée à partir du fichier. Cela fonctionne comme la fonction **scanf()** mais au lieu de lire les données de l'entrée standard, il lit les données du fichier. En fait, la plupart des arguments de la fonction **fscanf()** sont identiques à ceux de la fonction **scanf(),** à la différence qu'il a simplement besoin d'un argument supplémentaire, bien évidemment d'un pointeur de fichier. En cas de succès, cette fonction renvoie le nombre de valeurs lues et en cas d'erreur ou de fin du fichier, elle renvoie **EOF** ou -1.

Exemple 6 :

```
1
2  #include< stdio.h>
3  #include< stdlib.h>
4
5  int main(void)
6  {
7       File *f;
8       char nom[20];
9       int age;
10
11      f = fopen("test.txt", "r");
12
13      if(f==NULL){
14          printf("Erreur lors de l'ouverture d'un fichier");
15          exit(1);
16      }
17      // lire les données à partie de fichier
18      fscanf(f, "Nom : %s\tAge : %d\n", &nom, &age);
19
20      // afficher les données chargées
21      printf("Nom : %s\tAge : %d\n", nom, age);
22
23      fclose(f);
24
25      return 0;
26  }
27
```

24
25

4. fwrite et fread

Jusqu'à présent, nous utilisions le mode texte pour lire et écrire des données dans et à partir du fichier. Dans cette partie, nous allons apprendre à lire et à écrire des données dans et à partir du fichier en mode binaire. Rappelez-vous qu'en mode binaire, les données sont stockées dans le fichier de la même manière que dans la mémoire, de sorte qu'aucune transformation de données n'a lieu en mode binaire. En l'absence de transformation, le mode binaire est nettement plus rapide que le mode texte.

Les fonctions **fread()** et **fwrite()** sont couramment utilisées pour lire et écrire des données binaires dans et à partir du fichier, respectivement. Bien que nous puissions aussi les utiliser avec le mode texte.

fwrite

Syntaxe

```
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fp);
```

fwrite() permet d'écrire des données dans un fichier binaire
La fonction prend quatre arguments:

- adresse des données à écrire sur le fichier (**ptr**)
- taille des données à écrire sur le fichier (**size**)
- nombre de ce type de données(**n**)
- pointeur vers le fichier où vous voulez écrire (**fp**).

Pour mieux comprendre la fonction **fwrite()**, considérons les exemples suivants:

Exemple 7 : Ecrire une variable

```
int a=5;
fwrite(&a,sizeof(a),1,f); // f : pointeur vers le fichier
```

Exemple 8 : Ecrire un tableau

```
int tab[4]={2,5,7,8};
fwrite(tab,sizeof(tab),1,f); // f : pointeur vers le fichier
```

Exemple 9 : Ecrire une structure

```
struct etudiant{
    char nom[20];
    int age;
};
```

```
5
6struct etudiant etd={"mostafa",34};
7fwrite(&etd,sizeof(etd),1,f); // f : pointeur vers le fichier
```

Exemple 10 : Ecrire un tableau structures

```
?
1struct etudiant{
2    char nom[20];
3    int age;
4};
5
6struct etudiant etds[3]={{"mostafa",32},{"ismail",27},{"dounia",23}};
7fwrite(etds,sizeof(etds),1,f); // f : pointeur vers le fichier
```

Disons que nous ne souhaitons pas écrire tous les éléments du tableau dans le fichier. Nous souhaitons plutôt n'écrire que le premier et le deuxième éléments du tableau dans le fichier.

Exemple 11

```
?
1struct etudiant{
2    char nom[20];
3    int age;
4};
5
6struct etudiant etds[3]={{"mostafa",32},{"ismail",27},{"dounia",23}};
7
8fwrite(etds,sizeof(struct etudiant),2,f); // f : pointeur vers le fichier

`
```

fread

Syntaxe

```
?
1size_t fread(void *ptr, size_t size, size_t n, FILE *fp);
```

La fonction **fread()** est complémentaire de la fonction **fwrite().** La fonction **fread()** est couramment utilisée pour lire des données binaires. elle accepte les mêmes arguments que la fonction **fwrite().**

Exemple 13 : lire une variable

```
1int a=5;
2fread(a,sizeof(a),1,f); // f : pointeur vers le fichier
```

Exemple 14 : lire un tableau

```
 int tab[4]={2,5,7,8};
2fread(tab,sizeof(tab),1,f); // f : pointeur vers le fichier
```

Exemple 15 : lire une structure

```
struct etudiant{
    char nom[20];
    int age;
};

struct etudiant etd={"mostafa",34};
fread(&etd,sizeof(etd),1,f); // f : pointeur vers le fichier
```

Exemple 16 : lire un tableau de structres

```
struct etudiant{
    char nom[20];
    int age;
};

struct etudiant etds[3];
fread(&etds,sizeof(etds),5,f); // f : pointeur vers le fichier
```

Cette instruction lit les 5 premiers éléments de type **struct etudiant** du fichier et les stocke dans la variable **etds**.

Exemple 17:

```


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

int main(void)
{
    File *f;
    struct etudiant etd;
    f = fopen("test.txt", "r");

    if(f==NULL){
        printf("Erreur lors de l'ouverture d'un fichier");
        exit(1);
    }
    while( fread(&etd, sizeof(etd), 1, f) == 1 ){
        printf("Prénom : %s \n",etd.prenom);
        printf("age : %d",etd.age);

        // vider le tempon mémoire
        fflush(stdin);
    }

    fclose(f);

    return 0;
}


```