Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Informatique

TP ALGO1

Support TP programmation C - les fichiers -

Fait par :

Nom et prénom : ABDELMALEK BENMEZIANE

Contents

Chapter 1

Gestion des fichiers en C

1.1 Définition d'un fichier

1.1.1 Première définition

Ensemble organisé d'informations, désigné par un nom précis, que le système d'exploitation d'un ordinateur manipule comme une simple entité, dans sa mémoire ou sur un support de stockage.

1.1.2 deuxième définition

Un fichier fait référence à une source dans laquelle un programme stocke les informations/données sous forme d'octets séquentiels sur un disque (de manière permanente). Le contenu disponible sur un fichier n'est pas volatile comme la mémoire du compilateur en C.

1.2 Pourquoi avons-nous besoin de la gestion des fichiers en C ?

Il arrive parfois que la sortie générée par un programme après sa compilation et son exécution ne serve pas notre objectif. Dans de tels cas, nous pouvons vouloir vérifier la sortie du programme plusieurs fois. Désormais, compiler et exécuter le même programme plusieurs fois devient une tâche

fastidieuse pour tout programmeur. C'est exactement là que la gestion des fichiers devient utile.

Voyons quelques raisons pour lesquelles la gestion des fichiers facilite la programmation pour tous :

- Réutilisabilité : la gestion des fichiers nous permet de conserver les informations/données générées après l'exécution du programme.
- Gain de temps : certains programmes peuvent nécessiter une grande quantité de saisie de la part de leurs utilisateurs. Dans de tels cas, la gestion des fichiers vous permet d'accéder facilement à une partie d'un code à l'aide de commandes individuelles.
- Capacité de stockage remarquable : lorsque vous stockez des données dans des fichiers, vous n'avez plus à vous soucier de stocker toutes les informations en masse dans n'importe quel programme.
- Portabilité : le contenu disponible dans n'importe quel fichier peut être transféré vers un autre sans aucune perte de données dans le système informatique. Cela permet d'économiser beaucoup d'efforts et de minimiser le risque de codage erroné.

1.3 Types de fichiers dans un programme C

Lorsque nous parlons de gestion de fichiers, nous parlons de fichiers sous forme de fichiers de données. Or, ces fichiers de données sont disponibles sous 2 formes distinctes dans le langage C, à savoir :

- Text files
- Binary files

1.4 Les opérations/fonctions sur les fichiers en C

Nous pouvons utiliser une variété de fonctions pour ouvrir un fichier, le lire, écrire plus de données, créer un nouveau fichier, fermer ou supprimer un fichier, rechercher un fichier, etc. Ceux-ci sont connus sous le nom d'opérateurs de gestion de fichiers en C.

- fopen() : Ouverture d'un fichier existant ou un nouveau fichier
- fscanf() ou fgets() : Lecture à partir d'un fichier
- fprintf() ou fputs() : Écriture dans un fichier
- fseek(), rewind() : Déplacement vers un emplacement spécifique dans un fichier
- fclose() : Fermeture d'un fichier

1.5 Déclaration d'un fichier en C

La syntaxe du déclarartion d'un fichier en C est : FILE * fpointer; où fpointer est appelé le pointeur de fichier

1.5.1 Pointeur de fichier en C

Un pointeur de fichier est une référence à une position particulière dans le fichier ouvert. Il est utilisé dans la gestion des fichiers pour effectuer toutes les opérations sur les fichiers telles que la lecture, l'écriture, la fermeture, etc. Nous utilisons la macro FILE pour déclarer la variable de pointeur de fichier. La macro FILE est définie dans le fichier d'en-tête <stdio.h>.

1.6 Les opérations effectuées dans le traitement des fichiers

- Opening a file that already exists

- Creating a new file
- Reading content/ data from the existing file
- Writing more data into the file
- Deleting the data in the file or the file altogether

1.6.1 Ouverture d'un fichier

Pour ouvrir un fichier en C, la fonction fopen() est utilisée avec le nom du fichier ou le chemin du fichier ainsi que les modes d'accès requis.

La syntaxe

FILE * fopen(const char * nom_fichier, const char * mode_d'ouverture);

Les paramètres

- nom_fichier : nom du fichier lorsqu'il se trouve dans le même répertoire que le fichier source. Sinon, chemin d'accès complet.
- mode_d'ouverture : spécifie pour quelle opération le fichier est ouvert.

La valeur renvoyée

- Si le fichier est ouvert avec succès, renvoie un pointeur de fichier vers celui-ci.
- Si le fichier n'est pas ouvert, renvoie NULL.

Les modes d'ouverture

- w : Mode écriture (si le fichier n'existe pas, il va etre crée).
- r : Mode lecture.
- a : Mode ajouter à la fin de fichier.
- r+, w+ : à la fois écriture et lecture.
- a+ : à la fois ajouter a la fin et lecture

1.6.2 Fermeture d'un fichier

Une fois que nous avons écrit/lu un fichier dans un programme, nous devons le fermer (pour les fichiers binaires et texte). Pour fermer un fichier, nous utilisons la fonction fclose() dans un programme.

La syntaxe

fclose(file_pointer);

1.6.3 Lecture d'un fichier

Voici les fonctions pour la lecture d'un fichier

- int fscanf(FILE * ptr, const char * format, ...) : pour lire une chaîne (phrase) à partir d'un fichier (il lit un mot à partir du fichier).
- int fgetc(FILE * stream) : pour écrire un seul caractère du fichier.
- char * fgets(char * s, int n, FILE * stream) : la fonction lit une ligne de caractères à partir d'un fichier.

1.6.4 Ecriture dans un fichier

Voici les fonctions pour l'écriture dans un fichier

- int fprintf(FILE * fptr, const char * str, ...) : pour écrire une chaîne (phrase) dans un fichier (il envoie une sortie formatée vers un flux).
- int fputc(int c, FILE * stream) : pour écrire un seul caractère dans un fichier.
- int fputs(const char * s, FILE * stream) :écrit une ligne de caractères dans le fichier.

1.6.5 Gestion des erreurs

Lorsqu'une fonction est appelée en C, une variable nommée errno se voit automatiquement attribuer un code (valeur) qui peut être utilisé pour

identifier le type d'erreur rencontré. Il s'agit d'une variable globale indiquant l'erreur survenue lors de tout appel de fonction et définie dans le fichier d'en-tête errno.h.

Les fonctions de gestion des erreurs sont :

- void perror (const char *str)
- char *strerror (int errnum)

1.7 Exemples

1.7.1 Lecture d'un fichier caractère par caractère

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

int main(){
    /* File pointer to hold reference to our file */
    FILE *fPtr;
    char ch;

    /* Open file in r (read) mode */
    fPtr = fopen("data/file1.txt", "r");

    /* Check if file opening was successful */
    if (fPtr == NULL){
        /* Unable to open file, hence exit */
        perror("Unable to open file");
        exit(EXIT_FAILURE);
    }

    /* File open success message */
    printf("File opened successfully. Reading file contents
    character by character.\n\n");

    /* Read each character from the file and print */
    while ((ch = fgetc(fPtr)) != EOF){
```

```
        /* Print character read on console */
        printf("%c",ch);
    }

    fclose(fPtr);

    return 0;
}
```

1.7.2 Lecture d'un fichier line par line

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

#define BUFFER_SIZE 1000

int main(){
    /* File pointer to hold reference to our file */
    FILE *fPtr;

    char buffer[BUFFER_SIZE];
    int totalRead = 0;

    /* Open file in r (read) mode. */
    fPtr = fopen("data/file2.txt", "r");

    /* Check if file opening was successful */
    if (fPtr == NULL){
        /* Unable to open file hence exit */
        perror("Unable to open file");
        exit(EXIT_FAILURE);
    }

    /* File open success message */
    printf("File opened successfully.Reading file contents
    line by line. \n\n");

```

```
    /* Repeat this until read line is not NULL */
    while (fgets(buffer, BUFFER_SIZE, fPtr) != NULL){
        /* Total characters read count */
        totalRead = strlen(buffer);

        /* Trim new line character from last if exists */
        buffer[totalRead - 1] = (buffer[totalRead - 1] == '\n')
        ? '\0' : buffer[totalRead - 1];

        /* Print line read on console */
        printf("%s\n", buffer);
    }

    fclose(fPtr)

    return 0;
}
```

1.7.3 Ecriture d'un caractère dans un fichier

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

int main() {
    /* File pointer to hold reference to our file */
    FILE *fPtr;
    char ch;

    /* Demander a l'utilisateur d'entrer un caractere */
    printf("Entrez un caractere a ecrire dans le fichier: ");
    scanf("%c", &ch);

    /* Open file in write mode ("w") */
    fPtr = fopen("output_char.txt", "w");

    /* Check if file opening was successful */
    if (fPtr == NULL) {
```

```
        perror("Unable to open file");
        exit(EXIT_FAILURE);
    }

    /* Write a single character to the file */
    if (fputc(ch, fPtr) == EOF) {
        perror("Error writing character to file");
        if (fclose(fPtr) != 0) {
            perror("Error closing file after write failure");
        }
        exit(EXIT_FAILURE);
    }

    /* File write success message */
    printf("Character '%c' written to file successfully.\n", ch);

    /* Close the file */
    fclose(fPtr);

    return 0;
}
```

1.7.4 Ecriture d'une chaine de caractère dans un fichier

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

int main() {
    /* Pointeur de fichier pour detenir la r f rence du fichier*/
    FILE *fPtr;
    char str[1000];  // Tableau pour contenir la cha ne

    /* Demander    l'utilisateur d'entrer une cha ne */
    printf("Entrez une cha ne      crire   dans le fichier: ");
    fgets(str, sizeof(str), stdin);

    /* Ouvrir le fichier en mode   criture   ("w") */
```

```
16      fPtr = fopen("output_string.txt", "w");
17
18      /* V rifier si l'ouverture du fichier a r ussi */
19      if (fPtr == NULL) {
20          perror("Impossible d'ouvrir le fichier");
21          exit(EXIT_FAILURE);
22      }
23
24      /*  crire  la cha ne dans le fichier */
25      if (fputs(str, fPtr) == EOF) {
26          perror("Erreur lors de l' criture  de la cha ne
27          dans le fichier");
28          if (fclose (fPtr)!= 0){
29              perror ("Erreur lors de la fermeture du fichier
30              apr s une erreur d' criture ");
31          }
32          exit(EXIT_FAILURE);
33      }
34
35      printf("Cha ne "%s\"  crite  dans le fichier
36      avec succ s.\n", str);
37
38      /* Fermer le fichier */
39      fclose(fPtr);
40
41      return 0;
42  }
```

1.8 Les fichiers binaires

1.8.1 Les modes d'ouverture

- wb : Mode écriture (si le fichier n'existe pas, il va etre crée).
- rb : Mode lecture.
- ab : Mode ajouter à la fin de fichier.
- rb+, wb+ : à la fois écriture et lecture.

- ab+ : à la fois ajouter a la fin et lecture

1.8.2 Lecture

La Syntaxe

fread(address_of_Data, size_of_Data, numbers_of_Data, pointerToFile);

1.8.3 Ecriture

La Syntaxe

fwrite (address_of_Data, size_of_Data, numbers_of_Data, pointerToFile);

1.9 Exemples

1.9.1 Lecture depuis un fichier binaire

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

int main() {
    /* Pointeur de fichier pour detenir la r f rence du fichier*/
    FILE *fPtr;
    int arr[5];

    // Taille du tableau
    size_t arr_size = sizeof(arr) / sizeof(arr[0]);

    /* Ouvrir le fichier en mode binaire ("rb") */
    fPtr = fopen("output_binary.dat", "rb");

    /* V rifier si l'ouverture du fichier a r ussi */
    if (fPtr == NULL) {
        perror("Impossible d'ouvrir le fichier");
        exit(EXIT_FAILURE);
    }

```

```c
    /* Lire le tableau d'entiers depuis le fichier */
    size_t read_count = fread(arr, sizeof(int), arr_size, fPtr);

    /* V rifier si la lecture a r ussi */
    if (read_count != arr_size) {
        perror("Erreur lors de la lecture du fichier");
        fclose(fPtr);  // Fermer le fichier avant de quitter
        exit(EXIT_FAILURE);
    }

    /* Afficher les donn es lues */
    printf("Donn es lues depuis le fichier binaire:\n");
    for (size_t i = 0; i < arr_size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");

    /* Fermer le fichier */
    fclose(fPtr);

    return 0;
}
```

1.9.2 Ecriture dans un fichier binaire

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

int main() {
    /* Pointeur de fichier pour detenir la r f rence du fichier*/
    FILE *fPtr;
    int arr[] = {1, 2, 3, 4, 5};

    // Taille du tableau
    size_t arr_size = sizeof(arr) / sizeof(arr[0]);

    /* Ouvrir le fichier en mode binaire ("wb") */
    fPtr = fopen("output_binary.dat", "wb");
```

```c

    /* V rifier si l'ouverture du fichier a r ussi */
    if (fPtr == NULL) {
        perror("Impossible d'ouvrir le fichier");
        exit(EXIT_FAILURE);
    }

    /* Ecrire le tableau d'entiers dans le fichier */
    size_t write_count = fwrite(arr, sizeof(int), arr_size, fPtr);

    /* V rifier si l' criture  a r ussi */
    if (write_count != arr_size) {
        perror("Erreur lors de l' criture  dans le fichier");
        fclose(fPtr);  // Fermer le fichier avant de quitter
        exit(EXIT_FAILURE);
    }

    /* Message de succ s */
    printf("Tableau ecrit dans le fichier binaire avec succes.\n");

    /* Fermer le fichier */
    fclose(fPtr);

    return 0;
}
```

1.10 Les fonctions fseek() et rewind()

- fseek() : éplacement vers un emplacement spécifique dans un fichier.
- rewind() : éplacement vers le début de fichier.

1.11 Exemples

1.11.1 fseek()

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

```c

int main() {
    /* Pointeur de fichier pour detenir la r f rence du fichier*/
    FILE *fPtr;

    /* Ouvrir le fichier en mode binaire ("wb") */
    fPtr = fopen("example_seek.dat", "wb");

    /* V rifier si l'ouverture du fichier a r ussi */
    if (fPtr == NULL) {
        perror("Impossible d'ouvrir le fichier");
        exit(EXIT_FAILURE);
    }

    /*  crire  des donn es dans le fichier */
    const char *str1 = "Bonjour ";
    if (fwrite(str1, sizeof(char), 8, fPtr) != 8) {
        perror("Erreur lors de l' criture  des donn es");
        fclose(fPtr);
        exit(EXIT_FAILURE);
    }

    /* Utiliser fseek pour se d placer a la 5eme
    position (position 4) */
    if (fseek(fPtr, 4, SEEK_SET) != 0) {
        perror("Erreur lors de fseek");
        fclose(fPtr);
        exit(EXIT_FAILURE);
    }

    /*  crire  des donn es suppl mentaires apr s
    s' tre  d plac  dans le fichier */
    const char *str2 = "le monde!";
    if (fwrite(str2, sizeof(char), 9, fPtr) != 9) {
        perror("Erreur lors de l' criture  des
        donn es apr s fseek");
        fclose(fPtr);
        exit(EXIT_FAILURE);
    }
```

```
43
44      /* Message de succ s */
45      printf("Donn es  crites  dans le fichier avec fseek()\n");
46
47      /* Fermer le fichier */
48      fclose(fPtr);
49
50      return 0;
51  }
```

1.11.2 rewind()

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <errno.h>
4
5   int main() {
6       /* Pointeur de fichier pour detenir la r f rence du fichier*/
7       FILE *fPtr;
8
9       /* Ouvrir le fichier en mode  criture  ("w") */
10      fPtr = fopen("example_rewind.txt", "w");
11
12      /* V rifier si l'ouverture du fichier a r ussi */
13      if (fPtr == NULL) {
14          perror("Impossible d'ouvrir le fichier");
15          exit(EXIT_FAILURE);
16      }
17
18      /*  crire  des donn es dans le fichier */
19      const char *str = "Ceci est un test de rewind.";
20      if (fputs(str, fPtr) == EOF) {
21          perror("Erreur lors de l' criture  des donn es");
22          fclose(fPtr);
23          exit(EXIT_FAILURE);
24      }
25
26      /* Utiliser rewind pour remettre le curseur au debut du fichier*/
27      rewind(fPtr);
```

```c

    /* Lire et afficher le contenu du fichier a partir du d but*/
    char buffer[100];
    if (fgets(buffer, sizeof(buffer), fPtr) == NULL) {
        perror("Erreur lors de la lecture du fichier");
        fclose(fPtr);
        exit(EXIT_FAILURE);
    }

    /* Afficher les donn es lues */
    printf("Donn es lues apr s rewind : %s\n", buffer);

    /* Fermer le fichier */
    if (fclose(fPtr) != 0) {
        perror("Erreur lors de la fermeture du fichier");
        exit(EXIT_FAILURE);
    }

    return 0;
}
```