Lucrul cu fisiere

Lucrul cu fișiere este o componentă esențială a programării, mai ales când este necesară stocarea și prelucrarea persistentă a datelor. Limbajele C și C++ oferă suport extensiv pentru lucrul cu fișiere prin biblioteca standard **stdio.h** în C și **fstream** în C++.

1. Clasificarea fișierelor

În funcție de conținutul și modul de interpretare, fișierele sunt împărțite în două mari categorii:

a. Fișiere text

- Conțin date sub formă de caractere ASCII.
- Structura este organizată în linii, delimitate prin caracterul newline (\n).
- Citirea și scrierea se face în mod interpretat, ceea ce permite accesul direct la text uman-interpretabil.

b. Fișiere binare

- Conțin date în format brut (octeți).
- Nu există delimitatori între structuri de date; interpretarea este realizată la nivel de program.
- Sunt utilizate pentru stocarea eficientă a structurilor, obiectelor sau altor tipuri de date complexe.

2. Fluxuri de date (Streams)

Un flux reprezintă o abstracție a unei surse sau destinații de date. În C, fluxurile sunt manipulate prin intermediul bibliotecii <stdio.h>. Acestea pot fi de intrare (input) sau de ieșire (output), iar fiecare flux este asociat cu un fișier.

Fluxuri predefinite:

Flux	Descriere
stdin	Fluxul standard de intrare – în mod implicit, tastatura
stdout	Fluxul standard de ieșire – în mod implicit, ecranul
stderr	Fluxul standard de eroare – utilizat pentru mesaje de diagnostic

3. Etapele prelucrării unui fișier

a. Declararea și deschiderea fișierului

Pentru a lucra cu un fișier, se declară o variabilă de tip ***FILE*,*** care va conține informații legate de fișierul deschis:

```
FILE *fisier;
fisier = fopen("nume.txt", "r");
```

Funcția fopen are următoarea semnătură:

```
FILE *fopen(const char *nume_fisier, const char *mod_acces);
```

Moduri de acces:

Mod	Descriere
"r"	Citire; fișierul trebuie să existe deja
"w"	Scriere; fișierul este creat sau suprascris dacă există
"a"	Adăugare la finalul fișierului; creează fișierul dacă nu există
"r+"	Citire și scriere; fișierul trebuie să existe
"w+"	Scriere și citire; fișierul este creat sau suprascris
"a+"	Citire și adăugare la final
"t" / "b"	Specifică modul text sau binar; dacă este omis, este considerat text

b. Verificarea reușitei deschiderii

Este esențial să se verifice dacă fișierul a fost deschis cu succes:

```
if (fisier == NULL) {
    perror("Eroare la deschiderea fișierului");
    return 1;
}
```

c. Prelucrarea fișierului

Citire și scriere caracter cu caracter:

```
int fgetc(FILE *f);
int fputc(int c, FILE *f);
```

Citire și scriere linie cu linie:

```
char *fgets(char *sir, int dim, FILE *f);
int fputs(const char *sir, FILE *f);
```

d. Gestionarea poziției cursorului

Manipularea poziției curente în fișier se realizează cu ajutorul funcțiilor:

fseek(FILE *f, long offset, int whence) – repoziționează cursorul:
SEEK_SET: de la începutul fișierului
SEEK_CUR: față de poziția curentă
SEEK_END: față de sfârșitul fișierului
ftell(FILE *f) – întoarce poziția curentă a cursorului în octeți
rewind(FILE *f) – repoziționează cursorul la începutul fișierului

e. Detectarea sfârșitului de fișier

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

Această funcție returnează o valoare nenulă dacă s-a atins sfârșitul fișierului, altfel returnează 0.

f. Închiderea fișierului

După terminarea lucrului cu fișierul, este obligatorie eliberarea resurselor prin închiderea acestuia:

```
fclose(fisier);
```

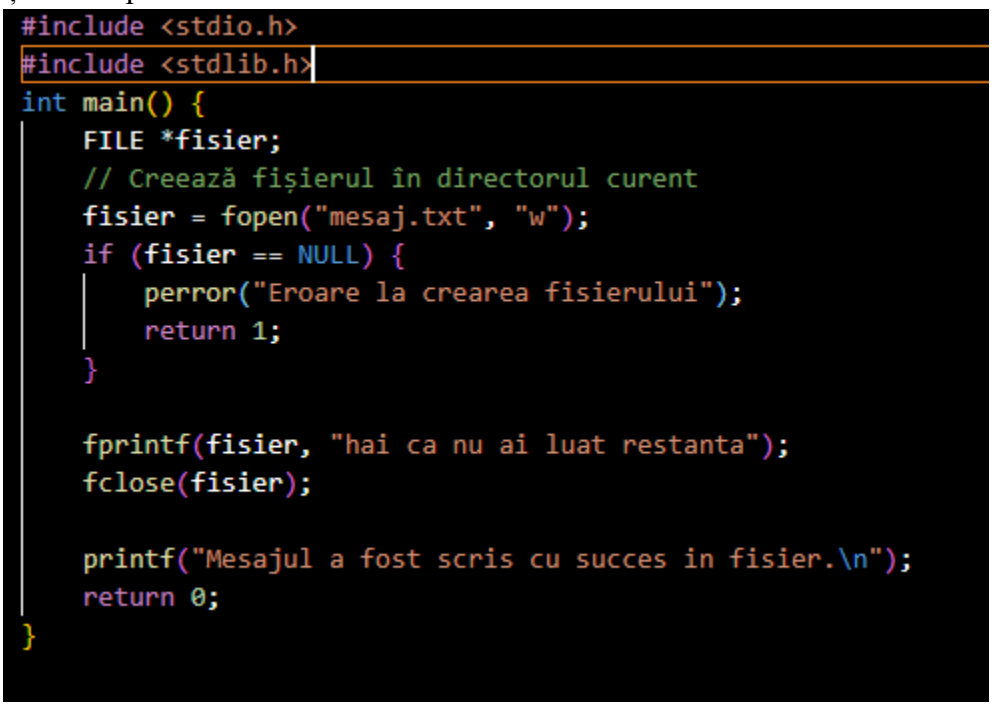
4. Particularități pentru fișiere binare

Pentru a manipula fișiere binare se folosesc funcțiile:

```
fread(void *ptr, size_t size, size_t nmemb, FILE *f);
fwrite(const void *ptr, size_t size, size_t nmemb, FILE *f);
```

Aceste funcții sunt eficiente pentru citirea și scrierea blocurilor de date (structuri, tablouri etc.).

1. Scrie un program C care creează fișierul mesaj.txt și scrie în el textul „hai ca nu ai luat restanta”, afișând un mesaj de confirmare dacă operația reușește sau un mesaj de eroare dacă fișierul nu poate fi deschis.

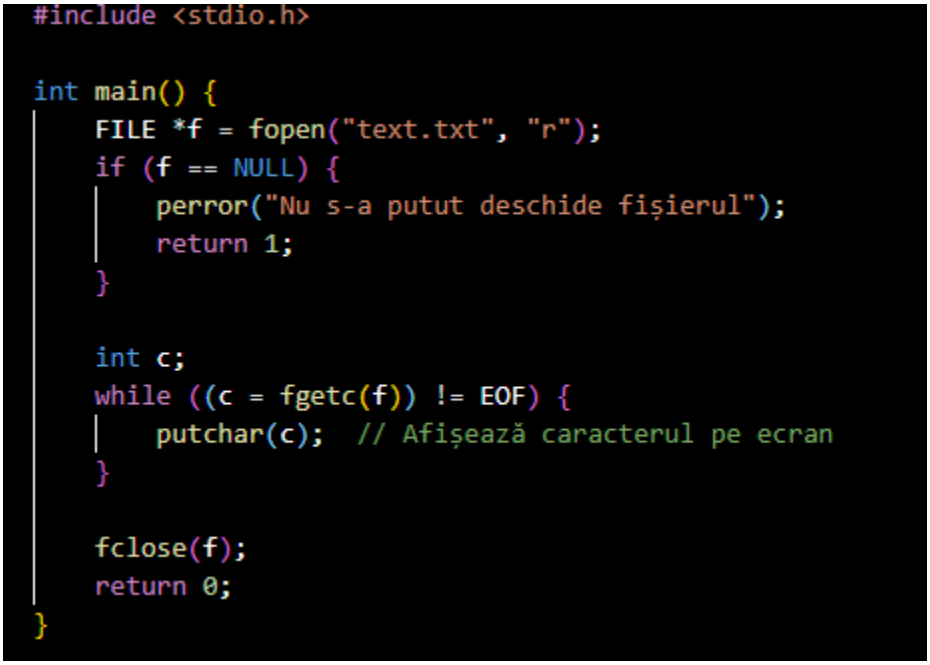


```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *fisier;
    // Creează fișierul în directorul curent
    fisier = fopen("mesaj.txt", "w");
    if (fisier == NULL) {
        perror("Eroare la crearea fisierului");
        return 1;
    }

    fprintf(fisier, "hai ca nu ai luat restanta");
    fclose(fisier);

    printf("Mesajul a fost scris cu succes in fisier.\n");
    return 0;
}
```

2. Scrie un program C care deschide fișierul text.txt pentru citire și afișează conținutul său caracter cu caracter pe ecran, tratând cazul în care fișierul nu poate fi deschis.



```c
#include <stdio.h>

int main() {
    FILE *f = fopen("text.txt", "r");
    if (f == NULL) {
        perror("Nu s-a putut deschide fișierul");
        return 1;
    }

    int c;
    while ((c = fgetc(f)) != EOF) {
        putchar(c);  // Afișează caracterul pe ecran
    }

    fclose(f);
    return 0;
}
```

3. Scrie un program C care deschide fișierul p.txt, citește conținutul său linie cu linie și afișează pe ecran numărul total de linii din fișier, tratând cazul în care fișierul nu poate fi deschis.

```c
#include <stdio.h>

int main() {
    FILE *f = fopen("p.txt", "r");
    if (f == NULL) {
        perror("Eroare la deschidere");
        return 1;
    }

    char linie[256];
    int numarLinii = 0;

    while (fgets(linie, sizeof(linie), f) != NULL) {
        numarLinii++;
    }

    printf("Fișierul conține %d linii.\n", numarLinii);
    fclose(f);
    return 0;
}
```

4. Scrie un program C care:

- Citește de la tastatură un număr întreg n (cu $n \leq 10$), reprezentând dimensiunea unei matrici pătratice $n \times n$;
- Citește de la tastatură toate cele $n \times n$ elemente întregi ale matricii;
- Scrie matricea într-un fișier text numit **matrice.txt**, astfel încât fiecare linie din fișier să corespundă unei linii din matrice;

```c
#include <stdio.h>
int main() {
    int n, i, j;
    int matrice[10][10];
    printf("Introdu dimensiunea matricei (n ≤ 10): ");
    scanf("%d", &n);
    printf("Introdu elementele matricii:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            scanf("%d", &matrice[i][j]);
        }
    }
```

```c
    // Scrierea în fișier
    FILE *f = fopen("matrice.txt", "w");
    if (f == NULL) {
        perror("Eroare la deschiderea fișierului pentru scriere");
        return 1;
    }
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            fprintf(f, "%d ", matrice[i][j]);
        }
        fprintf(f, "\n");
    }
    fclose(f);
    // Citirea din fișier și afișarea pe ecran
    f = fopen("matrice.txt", "r");
    if (f == NULL) {
        perror("Eroare la deschiderea fișierului pentru citire");
        return 1;
    }
    printf("\nMatricea citită din fișier este:\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            fscanf(f, "%d", &matrice[i][j]);
            printf("%d ", matrice[i][j]);
        }
        printf("\n");
    }
    fclose(f);
    return 0;
}
```

5. Scrie un program C care:

- Creează un fișier numit cuvinte.txt și scrie în el 3 cuvinte introduse de la tastatură, fiecare pe o linie separată;
- Apoi, cere utilizatorului să introducă un cuvânt de căutat;
- Deschide fișierul pentru citire și verifică dacă acel cuvânt există în fișier;
- Afișează un mesaj: „Cuvânt găsit!” sau „Cuvântul nu există în fișier.”.

```c
#include <stdio.h>
#include <string.h>
int main() {
    FILE *f;
    char cuvant[101], linie[101];
    int i, gasit = 0;

    // Deschidere fișier pentru scriere
    f = fopen("cuvinte.txt", "w");
    if (f == NULL) {
        printf("Eroare la crearea fișierului.\n");
        return 1;
    }
    // Citim 3 cuvinte și le scriem în fișier
    printf("Introdu 3 cuvinte (cate unul pe rand):\n");
    for (i = 0; i < 3; i++) {
        printf("Cuvant %d: ", i + 1);
        fgets(cuvant, sizeof(cuvant), stdin);
        cuvant[strcspn(cuvant, "\n")] = '\0';  // eliminăm \n
        fprintf(f, "%s\n", cuvant);
    }
    fclose(f);  // închidem fișierul după scriere
    // Cerem cuvântul de căutat
    printf("\nIntrodu cuvantul pe care vrei sa-l cauti in fisier: ");
    fgets(cuvant, sizeof(cuvant), stdin);
    cuvant[strcspn(cuvant, "\n")] = '\0';  // eliminăm \n

    // Deschidem fișierul pentru citire
    f = fopen("cuvinte.txt", "r");
    if (f == NULL) {
        printf("Eroare la deschiderea fișierului pentru citire.\n");
        return 1;
    }

    // Căutăm cuvântul linie cu linie
    while (fgets(linie, sizeof(linie), f)) {
        linie[strcspn(linie, "\n")] = '\0';  // eliminăm \n
        if (strcmp(cuvant, linie) == 0) {
            gasit = 1;
            break;
        }
    }

    fclose(f);
```

```
    if (gasit)
        printf("Cuvant gasit!\n");
    else
        printf("Cuvantul nu exista in fisier.\n");

    return 0;
}
```

6. Scrie un program C care:

- Creează un fișier text numit vector.txt și scrie în el un număr n (dimensiunea vectorului), urmat de n numere întregi, toate pe aceeași linie;
- Redeschide fișierul, citește valoarea lui n și apoi citește cele n elemente într-un vector alocat dinamic;
- Afișează vectorul pe ecran.

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

int main() {
    FILE *f;
    int n, i;
    int *vector;

    // Scriem vectorul în fișier
    f = fopen("vector.txt", "w");
    if (f == NULL) {
        printf("Eroare la crearea fișierului.\n");
        return 1;
    }

    // Exemplu: n = 5, valori: 10 20 30 40 50
    n = 5;
    fprintf(f, "%d ", n);
    for (i = 0; i < n; i++) {
        fprintf(f, "%d ", (i + 1) * 10);
    }

    fclose(f);

    // Citim vectorul din fișier
    f = fopen("vector.txt", "r");
    if (f == NULL) {
        printf("Eroare la deschiderea fișierului.\n");
```

```
        return 1;
    }

    fscanf(f, "%d", &n);  // citim dimensiunea

    // Alocare dinamică
    vector = (int *)malloc(n * sizeof(int));
    if (vector == NULL) {
        printf("Eroare la alocarea memoriei.\n");
        fclose(f);
        return 1;
    }

    // Citim elementele vectorului
    for (i = 0; i < n; i++) {
        fscanf(f, "%d", &vector[i]);
    }

    fclose(f);

    // Afișăm vectorul
    printf("Vectorul citit din fisier este:\n");
    for (i = 0; i < n; i++) {
        printf("%d ", vector[i]);
    }
    printf("\n");

    // Eliberăm memoria
    free(vector);

    return 0;
}
```