FIŞIERE TEXT ŞI BINARE

1. *Scrieţi o funcţie care preia numerele întregi dintr-un fişier text şi le scrie într-un fişier binar, respectiv o funcţie care preia numerele dintr-un fişier binar şi le scrie într-un fişier text.*

Rezolvare:

Funcţii de acest tip sunt necesare în programele care prelucrează informaţii stocate în fişiere binare, deoarece conţinutul lor este codificat sub forma unui şir de octeţi. Astfel, pentru a putea vizualiza datele dintr-un fişier binar într-un format specific unui tip de dată predefinit este necesară copierea lor într-un fişier text.

Funcţia `textBinar` copiază conţinutul unui fişier text într-un fişier binar. Astfel, funcţia primeşte ca argumente două şiruri de caractere prin care se specifică calea fişierului text `nfin`, respectiv calea fişierului binar `nfout`. Pentru a copia numerele întregi din fişierului text `nfin` în fişierul binar `nfout`, fişierul text este deschis în modul citire şi parcurs element cu element, folosind funcţia `fscanf`. Fişierul binar `nfout` este deschis în modul scriere, iar fiecare număr întreg citit din fişierul text este scris în fişierul binar folosind funcţia `fwrite`.

```
void textBinar(char *nfin, char *nfout)
{
    FILE *fin, *fout;
    int x;

    fin=fopen(nfin,"r");

    if(fin==NULL)
    {
        printf("Fisier inexistent!");
        return;
    }

    fout=fopen(nfout, "wb");

    while(fscanf(fin, "%d", &x)==1)
        fwrite(&x, sizeof(int), 1, fout);

    fclose(fin);
    fclose(fout);
}
```

Funcţia `binarText` scrie formatat conţinutul unui fişier binar într-un fişier text. Astfel, funcţia primeşte ca argumente două şiruri de caractere prin care se specifică calea fişierului binar `nfin`, respectiv calea fişierului text `nfout`. Pentru a scrie conţinutul fişierului binar în fişierul text, fişierul binar este deschis în modul citire şi apoi parcurs cu funcţia `fread`, în blocuri de octeţi având dimensiunea egală cu cea a unui întreg. Fişierul text `nfout` este deschis în modul scriere, iar fiecare număr întreg citit din fişierul binar este scris în fişierul text, folosind funcţia `fprintf`.

```
void binarText(char *nfin, char *nfout)
{
    FILE *fin, *fout;
    int x;

    fin=fopen(nfin,"rb");

    if(fin==NULL)
    {
        printf("Fisier inexistent!");
        return;
    }

    fout=fopen(nfout, "w");

    while(fread(&x, sizeof(int), 1, fin)==1)
        fprintf(fout, "%d ", x);

    fclose(fin);
    fclose(fout);

}
```

2. *Scrieți o funcție care să schimbe semnul fiecărui număr întreg dintr-un fișier binar. Funcția va avea ca parametru numele fișierului.*

Rezolvare:

Funcția `semnSchimbat` primește ca parametru numele fișierului sub forma unui șir de caractere. O metodă de rezolvare presupune parcurgerea fișierului cu metoda `fread`, în blocuri de octeți având dimensiunea egală cu cea a unui întreg, pentru a obține câte o valoare întreagă `x`. După ce semnul variabilei `x` este schimbat, noua valoare trebuie scrisă în fișierul de intrare. În acest sens, fișierul va fi deschis în modul mixt citire/scriere, respectiv în modul `rb+`.

Scrierea în fișier a valorii întregi `x` presupune, mai întâi, deplasarea pointer-lui de fișier înapoi cu `sizeof(int)` octeți față de poziția curentă, prin apelul `fseek(fin, -sizeof(int), SEEK_CUR)`, urmată de scrierea efectivă a valorii `x`.

```
void semnSchimbat(char* numeFisier)
{
      FILE* fin;
      int x;

      fin=fopen(numeFisier, "rb+");
      if(fin==NULL)
      {
          printf("Fisier inexistent!");
          return;
      }

      while(fread(&x, sizeof(int),1,fin)==1)
      {
```

```
            x=-x;
            fseek(fin, -sizeof(int), SEEK_CUR);
            fwrite(&x, sizeof(int),1,fin);
            fflush(stdin);
        }
      fclose(fin);
}
```

Codul de mai jos arată o modalitate de utilizare a funcţiilor definite în cadrul primelor două aplicaţii.

```
char fisierIn[50], fisierOut[50];

fgets(fisierIn, 50, stdin);
fisierIn[strlen(fisierIn)-1]='\0';

fgets(fisierOut, 50, stdin);
fisierOut[strlen(fisierOut)-1]='\0';

textBinar(fisierIn, fisierOut);
semnSchimbat(fisierOut);
binarText(fisierOut, fisierIn);
```

3. *Scrieţi un program care să creeze un fişier binar temporar format din valorile distincte dintr-un şir de n numere întregi citite de la tastatură şi apoi să exporte conţinutul fişierului binar într-un fişier text. Numele fişierului text se va citi de la tastatură.*

Rezolvare:

Algoritmul de extragere a valorilor distincte dintr-un vector cu n numere întregi care poate conţine duplicate presupune utilizarea unei structuri auxiliare de date de tip tablou care să reţină valorile din vectorul dat sub forma unei mulţimi (fără duplicate). În scopul de a nu utiliza un spaţiu suplimentar de memorie internă, putem utiliza un fişier temporar care să reţină toate valorile fără duplicate din vectorul dat. Un fişier temporar este creat pe suportul extern în momentul executării programului, iar după terminarea programului sau după închiderea fişierului, acesta va fi şters automat. Pentru a crea şi deschide un fişier temporar se utilizează funcţia `FILE *tmpfile(void)` din biblioteca `stdio.h`. Fişierul nou creat este deschis implicit în modul mixt `wb+`.

O metodă de rezolvare presupune, mai întâi, scrierea în fişierul binar a primei valori din vector prin apelul `fwrite(&v[0], sizeof(int), 1, fp)`. Ulterior se consideră pe rând toate valorile din vectorul dat şi se verifică dacă acestea au fost scrise deja în fişier. Acest lucru presupune o parcurgere repetată a fişierului temporar, deci pentru fiecare valoare căutată se poziţionează pointer-ul de fişier la începutul său prin apelul `fseek(fp, 0, SEEK_SET)`.

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

```
void distincte(int v[], int n, char *numeFisier)
{
    FILE *fp;
    fp = tmpfile();

    int i, x, g;

    fwrite(&v[0], sizeof(int), 1, fp);

    for(i=1; i<n ;i++)
    {
        fseek(fp, 0, SEEK_SET);
        g=0;

        while(fread(&x, sizeof(int), 1, fp)==1)
            if(x==v[i])
            {
                g=1;
                break;
            }

        if(g==0)
            fwrite(&v[i],sizeof(int), 1, fp);
    }

    fseek(fp, 0, SEEK_SET);

    fflush(fp);
    FILE *fout;

    fout=fopen(numeFisier, "w");

    while(fread(&x, sizeof(int), 1, fp)==1)
    {
        fprintf(fout, "%d ",x);
        printf("%d ",x);
    }

    fclose(fp);
    fclose(fout);
}

int main()
{
    int n, i;
    char numeFisier[50];

    printf("n = ");
    scanf("%d", &n);

    int v[n];

    for(i=0 ; i<n ;i++)
```

```
        scanf("%d", &v[i]);

    fgetc(stdin);
    printf("\nDati numele fisierului: ");
    fgets(numeFisier, 50, stdin);
    numeFisier[strlen(numeFisier)-1]='\0';

    distincte(v, n, numeFisier);


    return 0;
}
```

Variabila `g` a fost utilizată pentru marca faptul că o valoare din vectorul dat a fost găsită sau nu în fişierul temporar. Dacă valoarea curent analizată nu a fost găsită în fişierul temporar, atunci aceasta va fi scrisă după ultima valoare din fişier.

4. *Scrieţi un program care să genereze un număr aleator n format din exact două cifre şi apoi să realizeze următoarele operaţii:*

a) să construiască în memorie o matrice pătratică de dimensiune n, având forma indicată mai jos (pentru $n = 4$):

$$\begin{matrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{matrix}$$

b) să scrie matricea în fişierul binar matrice.bin;

c) să genereze două numere aleatoare i şi j cuprinse între 0 şi $n-1$ şi să interschimbe liniile i şi j din matrice direct în fişierul matrice.bin, fără a încărca matricea în memoria internă;

d) să exporte conţinutul fişierului binar în fişierul text matrice.txt.

Rezolvare:

Prima cerinţă presupune construirea în memorie a unei matrice cu n linii şi n coloane. Se observă faptul ca fiecare element $a[i][j]$ al matricei reprezintă valoarea absolută a diferenţei dintre cei doi indici i şi j.

```
int n, i, j;

printf("n= ");
scanf("%d", &n);

int a[n][n];

for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            a[i][j]=abs(i-j);
```

În scopul de a avea mai puţine accesări ale fişierului binar, se propune scrierea matricei, linie cu linie, ci nu element cu element.

```
FILE *fin;
fin=fopen("matrice.bin", "wb+");

for(i=0; i<n; i++)
        fwrite(a[i], n*sizeof(int), 1, fin);
```

Pentru a genera aleatoriu două numere naturale distincte cuprinse între 0 şi $n-1$ se propune utilizarea funcţiei `rand()`.

```
int l1,l2;
l1=rand()%n;
do
{
       l2=rand()%n;
}
while(l1==l2);
```

Cele două numere generate `l1` şi `l2` reprezintă numerele de ordine ale celor două linii pe care le vom interschimba. În acest sens, se poziţionează pointerul de fişier pe prima valoare de pe linia `l1`, folosind apelul `fseek(fin,n*l1*sizeof(int),SEEK_SET)` şi se copiază toate elementele liniei în tabloul `laux1`. Analog, se copiază toate elementele liniei `l2` în tabloul `laux2`. Ulterior, se va scrie în fişier tabloul `laux2` la poziţia dată de primul element de pe linia `l1`, respectiv tabloul `laux1` la poziţia data de primul element de pe linia `l2`.

```
int *laux1, *laux2;
fseek(fin,n*l1*sizeof(int),SEEK_SET);
laux1=(int*)malloc(n*sizeof(int));

fread(laux1,n*sizeof(int),1, fin);
fseek(fin,n*l2*sizeof(int),SEEK_SET);

laux2=(int*)malloc(n*sizeof(int));
fread(laux2,n*sizeof(int),1, fin);

fseek(fin,n*l1*sizeof(int),SEEK_SET);
fwrite(laux2, n*sizeof(int), 1, fin);

fseek(fin,n*l2*sizeof(int),SEEK_SET);
fwrite(laux1, n*sizeof(int), 1, fin);
```

Pentru realizarea ultimei cerinţe ne folosim de funcţia binarText de la exerciţiul 1. Mai jos este prezentat codul integral al aplicaţiei:

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

void binarText(char *nfin, char *nfout)//preluata de mai sus
```

```
int main()
{
    int n, i, j, *laux1, *laux2, l1,l2;

    printf("n= ");
    scanf("%d", &n);

    int a[n][n];

    for(i=0; i<n; i++)
        for(j=0; j<n; j++)
            a[i][j]=abs(i-j);

    for(i=0; i<n; i++)
    {
        for(j=0; j<n; j++)
            printf("%d ",a[i][j]);
        printf("\n");
    }

    FILE *fin;
    fin=fopen("matrice.bin", "wb+");

    for(i=0; i<n; i++)
        fwrite(a[i], n*sizeof(int), 1, fin);

    srand(time(NULL));

    l1=rand()%n;

    do
    {
       l2=rand()%n;
    }
    while(l1==l2);

    fseek(fin,n*l1*sizeof(int),SEEK_SET);
    laux1=(int*)malloc(n*sizeof(int));
    fread(laux1,n*sizeof(int),1, fin);

    fseek(fin,n*l2*sizeof(int),SEEK_SET);
    laux2=(int*)malloc(n*sizeof(int));
    fread(laux2,n*sizeof(int),1, fin);

    fseek(fin,n*l1*sizeof(int),SEEK_SET);
    fwrite(laux2, n*sizeof(int), 1, fin);

    fseek(fin,n*l2*sizeof(int),SEEK_SET);
    fwrite(laux1, n*sizeof(int), 1, fin);

    fclose(fin);
     binarText("matrice.bin", "matrice.txt");
```

```
    return 0;
}
```

5. *Definiţi o structură Student care să permită memorarea următoarelor informaţii despre un student: număr matricol, nume, grupa şi media. Numele se va păstra sub forma unui şir de caractere alocat dinamic. Implementaţi fiecare dintre următoarele cerinţe folosind câte o funcţie:*

a) încărcarea informaţiilor despre mai mulţi studenţi dintr-un fişier tip CSV (Comma-Separated Values) într-un tablou unidimensional;

b) salvarea conţinutului unui tablou unidimensional cu elemente de tip `Student` *într-un fişier binar;*

c) încărcarea conţinutului unui fişier binar într-un tablou unidimensional cu elemente de tip `Student`*;*

d) ştergerea din fişierul binar a unui student specificat prin codul său matricol;

e) modificarea în fişierul binar a mediei unui student specificat prin codul său matricol.

Rezolvare:

În primul rând, definim structura `Student` astfel:

```
typedef struct
{
    int nrMatricol;
    char *nume;
    int grupa;
    double medie;
} Student;
```

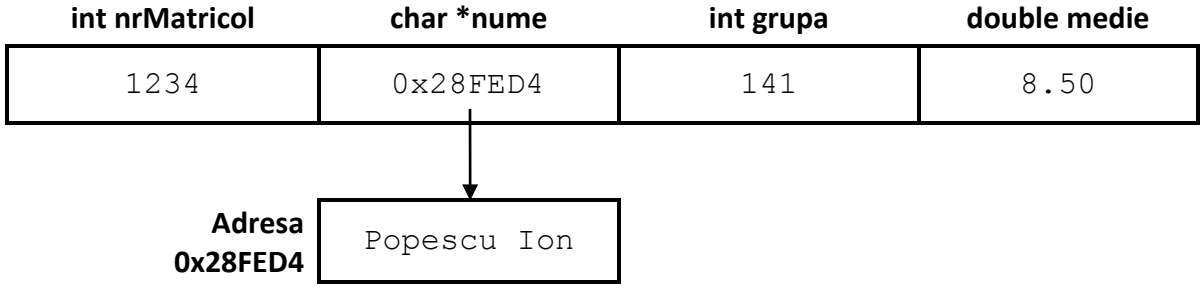
a) Fişierele CSV sunt fişiere text în care, pe fiecare linie, informaţiile sunt separate prin virgule. În cazul nostru, fiecare linie din fişier va conţine informaţii despre câte un student, astfel: *număr matricol*,*nume*,*grupa*,*media*. Pentru a încărca datele din fişierul CSV într-un tablou `ts` cu elemente de tip `Student`, vom citi fişierul CVS linie cu linie, după care vom separa informaţiile ţinând cont de faptul că separatorul este virgula. Parametrul `ns`, transmis prin adresă, va memora numărul elementelor tabloului `ts`.

```
void incarcareDateCSV(char *numeFisier, Student ts[], int *ns)
{
    char *p, linie[500];
    FILE *fin;

    fin = fopen(numeFisier, "r");

    if(fin == NULL)
    {
        printf("Fisier inexistent!");
        return;
    }

    *ns = 0;
    while(fgets(linie, 500, fin) != NULL)
    {
```

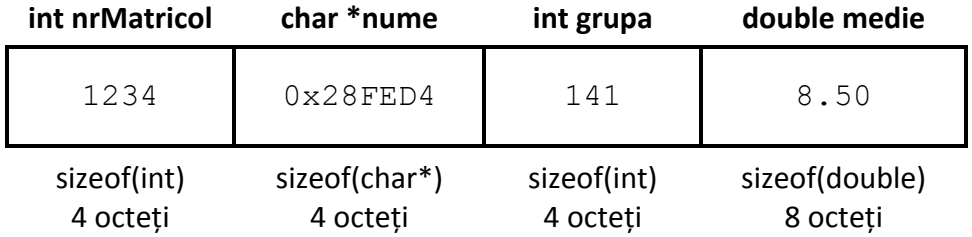
```
            p = strtok(linie, ",");
            ts[*ns].nrMatricol = atoi(p);

            p = strtok(NULL, ",");
            ts[*ns].nume = (char *)malloc(strlen(p) + 1);
            strcpy(ts[*ns].nume, p);

            p = strtok(NULL, ",");
            ts[*ns].grupa = atoi(p);

            p = strtok(NULL,",\n");
            ts[*ns].medie = atof(p);

            (*ns)++;
        }

        fclose(fin);
    }
```

Pentru a realiza conversia datelor numerice extrase din linia curentă sub forma unui şir de caractere în date numerice primitive, am folosit funcţiile `int atoi(char *șir)` pentru numere întregi şi funcţia `double atof(char *șir)` pentru numere reale, ambele incluse în biblioteca `stdlib.h`.

b) Un element al unui tablou unidimensional `ts` cu elemente de tip `Student` are următoarea formă:



Se observă faptul că în câmpul `nume`, alocat dinamic, nu este memorat efectiv şirul de caractere care conţine numele unui student, ci adresa sa de început din zona de memorie heap. Din acest motiv, dacă am scrie direct conţinutul tabloului `ts` într-un fişier binar, am salva în fişierul binar nişte adrese de memorie în locul numelor studenţilor respectivi, aşa cum se poate observa în figura de mai jos:



Ulterior, în momentul în care am redeschide fișierul binar și am citi informațiile despre studenți din el, s-ar încărca adresele respective, dar la acele adrese s-ar găsi alte informații (posibil aparținând altor programe sau chiar sistemului de operare)!
Din acest motiv, trebuie să scriem în fișierul binar, pe rând, valorile câmpurilor fiecărui element `ts[i]` al tabloului, iar în cazul câmpului `nume`, care are o lungime variabilă, vom scrie în fișierul binar mai întâi dimensiunea sa în octeți, respectiv `strlen(ts[i])+1`, după care vom scrie șirul de caractere corespunzător, astfel:

int nrMatricol	**char *nume**		**int grupa**	**double medie**
1234	12	Popescu Ion	141	8.50
sizeof(int) 4 octeți	sizeof(int) 4 octeți	strlen(nume)+1 12 octeți	sizeof(int) 4 octeți	sizeof(double) 8 octeți

Implementarea funcției este următoarea:

```
void scriereDateBinar(Student ts[], int n, char *numeFisier)
{
    FILE* fout;
    unsigned int i, dim_nume;

    fout = fopen(numeFisier, "wb");

    for(i = 0; i < n; i++)
    {
        fwrite(&ts[i].nrMatricol,sizeof(ts[i].nrMatricol), 1, fout);

        dim_nume = strlen(ts[i].nume) + 1;
        fwrite(&dim_nume, sizeof(dim_nume), 1, fout);
        fwrite(&ts[i].nume, dim_nume, 1, fout);

        fwrite(&ts[i].grupa, sizeof(ts[i].grupa), 1, fout);
        fwrite(&ts[i].medie, sizeof(ts[i].medie), 1, fout);
    }

    fclose(fout);
}
```

c) Încărcarea conținutului unui fișier binar într-un tablou unidimensional cu elemente de tip `Student` se va realiza prin operații de citire similare celor de scriere din funcția anterioară:

```
void incarcareTablouBinar(char *numeFisier, Student ts[], int *n)
{
    FILE* fin;
    unsigned int i, dim_nume;
```

```
    int auxm;

    fin = fopen(numeFisier, "rb");

    if(fin == NULL)
    {
        printf("Fisier inexistent!");
        return;
    }

    i = 0;
    while(fread(&auxm, sizeof(auxm), 1, fin) == 1)
    {
        ts[i].nrMatricol = auxm;
        fread(&dim_nume, sizeof(dim_nume), 1, fin);
        ts[i].nume = (char *)malloc(dim_nume);
        fread(&ts[i].nume, dim_nume, 1, fin);

        fread(&ts[i].grupa, sizeof(ts[i].grupa), 1, fin);
        fread(&ts[i].medie, sizeof(ts[i].medie), 1, fin);
        i++;
    }

    *n = i;

    fclose(fin);
}
```

După utilizarea unui tablou unidimensional `tp` cu elemente de tip Student, este necesară eliberarea zonelor de memorie alocate dinamic pentru numele studenților!

d) Funcția `modificareStudent` va avea ca parametri, pe lângă numele fișierului binar în care sunt salvate informațiile despre studenți, numărul matricol `nrm` al studentului și noua sa medie `medn`. Funcția va întoarce valoarea `1` dacă modificarea a putut fi efectuată (în fișierul binar există informații despre studentul respectiv) sau `0` în caz contrar. Deoarece căutarea unui student se efectuează după numărul său matricol, nu vom citi din fișierul binar toate informațiile despre fiecare student, ci doar numărul său matricol `nrm_crt` și lungimea numelui `dim_nume`. Astfel, în momentul în care vom găsi în fișierul binar numărul matricol căutat `nrm`, putem actualiza direct valoarea mediei studentului respectiv, deoarece știm faptul că aceasta este memorată peste `dim_nume+sizeof(int)` octeți la dreapta față de poziția curentă. Dacă numărul matricol curent `nrm_crt` nu este cel căutat, atunci vom avansa la următorul număr matricol, adică peste `dim_nume+ sizeof(int)+sizeof(double)` octeți la dreapta față de poziția curentă.

```
int modificareStudent(char *numeFisier, int nrm, double medn)
{
    FILE* fin;
    unsigned int dim_nume;
    int nrm_crt;
```

```
    fin = fopen(numeFisier, "rb+");

    if(fin == NULL)
    {
        printf("Fisier inexistent!!!");
        return 0;
    }

    while(fread(&nrm_crt, sizeof(int), 1, fin ) == 1)
    {
        fread(&dim_nume, sizeof(dim_nume), 1, fin);

        if(nrm_crt == nrm)
        {
            fseek(fin, dim_nume + sizeof(int), SEEK_CUR);
            fwrite(&medn, sizeof(medn), 1, fin);
            fclose(fin);
            return 1;
        }

        fseek(fin, dim_nume + sizeof(int) + sizeof(double), SEEK_CUR);
    }

    fclose(fin);

    return 0;
}
```

e) Funcţia `stergereStudent` va avea ca parametri, pe lângă numele fişierului binar în care sunt salvate informaţiile despre studenţi, şi numărul matricol `nrm` al studentului ale cărui informaţii dorim să le ştergem. Funcţia va întoarce valoarea `1` dacă ştergerea a putut fi efectuată cu succes (în fişierul binar există informaţii despre studentul respectiv) sau `0` în caz contrar.

Deoarece în limbajul C nu există funcţii predefinite care să permită trunchierea unui fişier, operaţia de ştergere se poate implementa astfel:
1. se copiază selectiv informaţiile din fişierul respectiv într-un fişier auxiliar;
2. se şterge efectiv fişierul iniţial de pe suportul de memorie externă;
3. se redenumeşte fişierul auxiliar cu numele fişierului iniţial.

Căutarea studentului ale cărui informaţii dorim să le ştergem din fişierul binar se va efectua după numărul său matricol `nrm`. Dacă numărul matricol curent `nrm_crt` este diferit de `nrm`, atunci vom citi toate informaţiile referitoare la studentul respectiv şi le vom scrie în fişierul binar auxiliar. În caz contrar, vom ignora informaţiile respective, prin mutarea pointer-ului de fişier înainte cu `dim_nume+sizeof(int)+sizeof(double)` octeţi.

```
int stergereStudent(char *numeFisier, int nrm)
{
    FILE *fin, *fout;
    unsigned int dim_nume;
```

```
    int nrm_crt, gs;
    char aux[101];

    fin = fopen(numeFisier, "rb");
    fout = fopen("tmp_studenti.bin", "wb");

    if(fin == NULL)
    {
        printf("Fisier inexistent!\n");
        return 0;
    }

    gs = 0;
    while(fread(&nrm_crt, sizeof(int), 1, fin ) == 1)
    {
        fread(&dim_nume, sizeof(dim_nume), 1, fin);

        if(nrm_crt != nrm)
        {
            fread(aux, dim_nume+sizeof(int)+sizeof(double), 1, fin);
            fwrite(&nrm_crt, sizeof(nrm_crt), 1, fout);
            fwrite(&dim_nume, sizeof(dim_nume), 1, fout);
            fwrite(aux, dim_nume+sizeof(int)+sizeof(double), 1, fout);
        }
        else
        {
            gs = 1;
            fseek(fin, dim_nume+sizeof(int)+sizeof(double), SEEK_CUR);
        }
    }

    fclose(fin);
    fclose(fout);

    remove(numeFisier);
    rename("tmp_studenti.bin", numeFisier);

    return gs;
}
```

Pentru ştergerea fizică a unui fişier se foloseşte funcţia `int remove(const char *cale_fisier)` din biblioteca `stdio.h`, iar pentru redenumirea unui fişier funcţia `int rename(const char *cale_fisier, const char *nume_nou)` din aceeaşi bibliotecă. Ambele funcţii întorc valoarea 0 în cazul în care operaţia respectivă se efectuează cu succes sau o valoare nenulă în caz contrar.

6. În momentul copierii sau transmiterii printr-o reţea a unui fişier pot să apară diverse erori cauzate de modificarea, intenţionată sau nu, a valorilor unor octeţi. Unul dintre mecanismele utilizate pentru a verifica integritatea unui fişier îl constituie calcularea unei *sume de control* (*checksum*) pentru fişierul respectiv, folosind un anumit algoritm ce antrenează toţi octeţii fişierului. Valoarea astfel obţinută este transmisă/publicată, pe un canal de comunicaţie, alături de fişierul respectiv. Verificarea integrităţii fişierului

recepționat se poate realiza prin calcularea sumei de control a acestuia, folosind același algoritm, urmată de compararea rezultat obținut cu suma de control publicată pentru fișierul respectiv.

Unul dintre cei mai simpli algoritmi de calcul a unei sume de control constă în adunarea tuturor valorilor octeților din fișier (considerați ca fiind fără semn și, eventual, grupați câte 2, 4 sau 8) și reducerea modulo o anumită constantă a sumei obținute (de obicei, constanta este egală cu 2^b, unde b semnifică numărul de biți pe care se reprezintă tipul de date întreg fără semn utilizat pentru calcularea sumei). De exemplu, dacă un fișier binar este format din octeții fără semn având valorile 200, 7, 153, 9, 110, 88 vom obține suma $s = 200 + 7 + +153 + 9 + 110 + 88 = 567$. Reducând suma s modulo 256 (deoarece $b = 8$), vom obține suma de control 55. Se observă cu ușurință faptul că această metodă nu permite detectarea mai multor tipuri de modificări care se pot efectua asupra octeților unui fișier, cum ar fi schimbarea ordinii octeților sau înlocuirea unui grup de octeți cu un alt grup având aceeași suma a valorilor.

Un algoritm mai sensibil la modificarea octeților a fost propus în 1982 de către John Flechter. O descriere a acestui algoritm o puteți găsi în pagina https://en.wikipedia.org/wiki/Fletcher%27s_checksum. Algoritmul constă în parcurgerea fișierului binar în grupuri de câte B octeți fără semn ($B \in \{1,2,4\}$) și calculul a două sume, ambele reduse modulo $2^{8*B} - 1$, astfel:

- termenii primei sume (`suma1`) sunt grupurile de câte B octeți citite din fișierul binar;
- termenii celei de-a doua sume (`suma2`) sunt valorile obținute pentru prima sumă (`suma1`) după fiecare grup de octeți citit.

Suma de control (`cs`) se obține prin concatenarea reprezentării binare a valorii celei de-a doua sume (`suma2`) cu reprezentarea binară a primei sume (`suma1`). Astfel, suma de control trebuie să fie calculată utilizând un tip de date întreg fără semn având o lungime a reprezentării binare egală cu $2 * B$!

De exemplu, pentru șirul de octeți de mai sus, considerând $B = 1$ și modulul $2^8 - 1 = 255$, vom calcula suma de control astfel:

Octet curent	**suma1**	**suma2**
–	0	0
200	200 % 255 = 200	200 % 255 = 200
7	(200 + 7) % 255 = 207	(200 + 207) % 255 = 152
153	(207 + 153) % 255 = 105	(152 + 105) % 255 = 2
9	(105 + 9) % 255 = 114	(2 + 114) % 255 = 116
110	(114 + 110) % 255 = 224	(116 + 224) % 255 = 85
88	(224 + 88) % 255 = 57	(85 + 57) % 255 = 142
Suma de control	$\overbrace{10001110}^{142}\overbrace{00111001}^{57} = 36409$	

În continuare, este prezentat programul complet pentru calculul sumei de control folosind algoritmul lui Flechter în care am considerat $B = 2$ octeţi, deci se va utiliza modulul $2^{16} - 1 = 65535$, iar suma de control se va reprezenta pe 32 de biţi fără semn:

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

int main()
{
    FILE *f;

    unsigned short int w, suma1, suma2;
    unsigned char i, x;
    unsigned int dimf, cs;
    unsigned char numef[101];

    printf("Numele fisierului: ");
    fgets(numef, 101, stdin);
    numef[strlen(numef) - 1] = '\0';

    f = fopen(numef, "rb");

    fseek(f, 0, SEEK_END);
    dimf = ftell(f);
    rewind(f);

    suma1 = suma2 = 0;
    while(fread(&w, sizeof(unsigned short int), 1, f) == 1)
    {
        suma1 = (suma1 + w) % 65535;
        suma2 = (suma2 + suma1) % 65535;
    }

    if(dimf % 2 != 0)
    {
        fread(&x, 1, 1, f);
        suma1 = (suma1 + x) % 65535;
        suma2 = (suma2 + suma1) % 65535;
    }

    fclose(f);

    cs = suma2;
    cs = cs << 16 | suma1;

    printf("\n\nSuma de control: %u\n", cs);

    return 0;
}
```

Deoarece fişierul binar este parcurs în blocuri de câte 2 octeţi, în cazul în care dimensiunea fişierului în octeţi este un număr impar, atunci ultimul octet trebuie citit separat.

Se poate observa faptul că algoritmul lui Flechter calculează o sumă de control cu ajutorul căreia pot fi detectate modificări ale ordinii octeţilor sau înlocuiri ale unui grup de octeţi cu un alt grup având aceeaşi suma a valorilor. Totuşi, există şi câteva modificări pe care acest tip de sumă de control nu le poate detecta. Puteţi să daţi exemplu de o astfel de modificare?

Probleme propuse

1. Scrieţi un program care să creeze un fişier binar cu numere întregi generate aleator într-un interval [-n, n], cu n citit de la tastatură, după care să modifice fiecare număr din fişier astfel: numerele pare se înjumătăţesc, iar cele impare se dublează.

2. Scrieţi un program care să creeze un fişier binar cu numere întregi generate aleator într-un interval [-n, n], cu n citit de la tastatură,, după care să sorteze crescător numerele din fişier fără a utiliza structuri de date auxiliare şi apoi să afişeze pe ecran valoarea maximă din fişier, precum şi frecvenţa sa de apariţie.

3. Scrieţi câte o funcţie pentru rezolvarea următoarelor cerinţe:
   a) salvarea în fişierul binar *matrice.bin* a unei matrice pătratice de dimensiune $n \times n$, având elementele citite de la tastatură;
   b) determinarea numărului de linii/coloane ale unei matrice pătratice salvate în fişierul *matrice.bin*;
   c) exportarea conţinutul fişierului binar în fişierul text *matrice.txt*;
   d) afişarea pe ecran a elementelor aflate pe diagonala principală a unei matrice salvate în fişierul *matrice.bin*, fără utilizarea unor structuri de date auxiliare.

4. Definiţi o structură *Angajat* care să permită memorarea următoarelor informaţii despre un angajat: *cnp, nume*, *vârstă* şi *salariu*. Numele se va păstra sub forma unui şir de caractere alocat dinamic. Scrieţi câte o funcţie pentru rezolvarea fiecăreia dintre următoarele cerinţe:
   a) scrierea în fişierul binar *angajati.bin* a informaţiilor despre mai mulţi angajaţi cu datele citite de la tastatură;
   b) afişarea pe ecran a datelor despre un angajat din fişierul binar *angajati.bin* pentru care se cunoaşte codul numeric personal;
   c) adăugarea unui nou angajat în fişierul binar *angajati.bin*;
   d) scrierea în fişierul text *salarii.txt* a numelui şi salariului fiecărui angajat din fişierul binar *angajati.bin*, precum şi salariul mediu;
   e) afişarea pe ecran a numelor angajaţilor din fişierul binar *angajati.bin* care au salariul egal cu cel maxim ;
   f) ştergerea unui angajat din fişierul binar *angajaţi.bin* pentru care se cunoaşte codul numeric personal.

5. Scrieţi o funcţie care să încarce într-un tablou unidimensional conţinutul unui fişier binar în care sunt memorate numere întregi, să sorteze descrescător tabloul respectiv folosind funcţia `qsort` din `stdlib.h` şi apoi să-l afişeze pe ecran. Funcţia va avea ca parametru numele fişierului binar.

6. O imagine color poate fi transformată într-o imagine sepia înlocuind valorile (R, G, B) ale fiecărui pixel cu valorile (R', G', B') definite astfel:

$$\begin{aligned} R' &= \min\{[0.393 * R + 0.769 * G + 0.189 * B], 255\} \\ G' &= \min\{[0.349 * R + 0.686 * G + 0.168 * B], 255\} \\ B' &= \min\{[0.272 * R + 0.534 * G + 0.131 * B], 255\} \end{aligned}$$

   unde prin $[x]$ am notat partea întreagă a numărului real x. Scrieţi un program care să citească de la tastatură calea unei imagini color de tip bitmap şi să o transforme într-o imagine sepia.

7. O îmbunătăţire a algoritmului lui Flechter pentru calculul unei sume de control (prezentat în aplicaţia 6) a fost propusă în anul 1995 de către Mark Adler. O descriere a acestui algoritm o puteţi găsi în pagina https://en.wikipedia.org/wiki/Adler-32. Scrieţi un program care să implementeze acest algoritm pentru un fişier binar.