Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

**RAPORT**

Lucrare de laborator Nr.3

*la Structuri de Date și Algoritmi*

Tema: Algoritmi de prelucrare a fișierelor text

A efectuat: st. gr. SI-212 Șeremet Alexandru

A verificat: lect. asist. Mititelu Vitalii

Chişinău 2022

**Scopul:** Programarea algoritmilor de prelucrare a fișierelor text prin

utilizarea funcțiilor, pointerilor, alocării dinamice a memoriei în

limbajul C.

**Sarcina:** Pentru fișierele text de efectuat sarcina specificată în varianta de mai jos:

1. Crearea unui fișier binar nou, introducerea câmpurilor structurilor de

la tastatură și înscrierea lor în acest fișier.

2. Afișarea elementelor fișierului binar la ecran.

3. Adăugarea unei structuri noi la sfârșit de fișier.

4. Modificarea unei structuri a fișierului.

5. Căutarea după un câmp al structurii în fișier.

6. Sortarea structurilor fișierului după un careva câmp.

7. Eliminarea unei structuri din fișier.

8. Ștergerea fișierului de pe disc (la dorință)

0. Ieșire din program.

Să se elaboreze funcțiile pentru implementarea opțiunilor meniului.

**Varianta 15:**

Se dă un fișier care conține pe fiecare rând un nume (un cuvânt) urmat de 5 note.

Se cere să se construiască un al doilea fișier care conține numele urmat de medie,

ordonat descrescător după medie. Dacă un nume apare pe mai multe linii in

fișierul original, liniile respective vor fi scrise într-un fișier cu numele 'eronat', și

pentru acestea nu se va calcula media.

**Rezumat:**

**Introducere**

Un fişier este o structură dinamică, situată în memoria secundară (pe disk-uri). Limbajul C permite operarea cu fişiere:

de tip text - un astfel de fişier conţine o succesiune de linii, separate prin new line ('\n')

de tip binar - un astfel de fişier conţine o succesiune de octeti, fără nici o structură.

Prelucrarea unui fişier presupune asocierea acestuia cu un canal de I/E (numit flux sau stream). Există trei canale predefinite, care se deschid automat la lansarea unui program:

stdin - fişier de intrare, text, este intrarea standard - tastatura

stdout - fişier de iesire, text, este ieşirea standard - ecranul monitorului.

stderr – fişier de iesire, text, este ieşirea standard unde sunt scris mesajele de eroare - ecran.

**Pentru a prelucra un fişier, trebuie parcurse următoarele etape:**

se defineşte o variabilă de tip FILE\* pentru accesarea fişierului; FILE este un tip structură definit în <stdio.h>, care conţine informaţii referitoare la fişier şi la tamponul de transfer de date între memoria centrală şi fişier (adresa, lungimea tamponului, modul de utilizare a fişierului, indicator de sfârsit, de poziţie în fişier). Puteți citi mai multe aici .

se deschide fişierul pentru un anumit mod de acces, folosind funcţia de bibliotecă fopen, care realizează şi asocierea între variabila fişier şi numele extern al fişierului

se prelucrează fişierul în citire/scriere cu funcţiile specifice

se închide fişierul folosind funcţia de bibliotecă fclose

**Funcții**

Mai jos se prezintă restul funcţiilor de prelucrare a fişierelor. Pentru documentația oficială puteți citi aici.

fopen

FILE \*fopen(const char \*filename, const char \*mod);

deschide fişierul cu numele filename pentru acces de tip mod.

Returnează pointer la fişier sau NULL dacă fişierul nu poate fi deschis; valoarea returnată este memorată în variabila fişier, care a fost declarată pentru accesarea lui.

Modul de deschidere poate fi:

“r” - readonly , este permisă doar citirea dintr-un fişier existent

“w” - write, crează un nou fişier, sau dacă există deja, distruge vechiul continut

“a” - append, deschide pentru scriere un fişier existent ( scrierea se va face în continuarea informaţiei deja existente în fişier, deci pointerul de acces se plasează la sfârşitul fişierului )

“+” - permite scrierea şi citirea - actualizare (ex: “r+”, “w+”, “a+”). Între read şi write trebuie repoziţionat cursorul de acces printr-un apel la fseek.

“b” - specifică fişier de tip binar

“t” - specifică fişier de tip text (implicit), la care se face automat conversia CR-LF(“\n\f”) în sau din CR ('\n').

fclose

int fclose(FILE \*pFile);

închide fişierul asociat cu variabila pFile şi eliberează zona tampon; returnează 0 la succes, EOF (end of file) la eroare

fseek

int fseek(FILE \*pFile, long offset, int whence);

repoziţionează pointerul asociat fişierului pFile; offset - numărul de octeţi între poziţia dată de whence şi noua poziţie.

whence - are una din cele trei valori posibile:

SEEK\_SET = 0 - Căutarea se face de la începutul fişierului

SEEK\_CUR = 1 - Căutare din poziţia curentă

SEEK\_END = 2 - Căutare de la sfârşitul fişierului

ftell

long ftell(FILE \*pFile);

întoarce poziţia curentă în cadrul fișierului asociat cu pFile.

fgetpos

int fgetpos(FILE \*pFile, fpos\_t \*ptr);

această funcţie memorează poziţia curentă în variabila ptr în cadrul fişierului asociat cu pFile (ptr va putea fi folosit ulterior cu funcţia fsetpos).

fsetpos

int fsetpos(FILE \*pFile, const fpos\_t \*ptr);

această funcţie setează poziţia curentă în fişierul asociat cu pFile la valoarea ptr, obţinută anterior prin funcţia fgetpos.

feof

int feof(FILE \*fis);

returnează 0 dacă nu s-a detectat sfârşit de fişier la ultima operaţie de citire, respectiv o valoare nenulă (adevărată) pentru sfârşit de fişier.

freopen

FILE\* freopen(const char \*filename, const char \*mode, FILE \*fp);

se închide fişierul fp, se deschide fişierul cu numele filename în modul mode şi acesta se asociază la fp; se întoarce fp sau NULL în caz de eroare.

fflush

int fflush(FILE \*fp);

Această funcţie se utilizează pentru fişierele deschise pentru scriere şi are ca efect scrierea în fişier a datelor din bufferul asociat acestuia, care înca nu au fost puse în fişier.

Citirea şi scrierea în/din fişiere

Citirea/scrierea în fişiere se poate face în două moduri (în funcție de tipul fişierului): în mod text sau în mod binar. Principalele diferenţe dintre cele două moduri sunt:

în modul text, la sfarsitul fişierului se pune un caracter suplimentar, care indică sfârşitul de fişier. În DOS şi Windows se utilizează caracterul cu codul ASCII 26 (Ctrl-Z), iar în Unix se utilizează caracterul cu codul ASCII 4. Dacă citim un fişier în mod text, citirea se va opri la intâlnirea acestui caracter, chiar dacă mai există şi alte caractere după el. În modul binar nu există caracter de sfârşit de fişier (mai precis, caracterul cu codul 26, respectiv 4, este tratat la fel ca şi celelalte caractere).

în DOS şi Windows, în modul text, sfârşitul de linie este reprezentat prin două caractere, CR (Carriage Return, cod ASCII 13) şi LF (Line Feed, cod ASCII 10). Atunci când în modul text scriem un caracter '\n' (LF) în fişier, acesta va fi convertit într-o secventă de 2 caractere CR şi LF. Când citim în mod text dintr-un fişier, secvenţa CR, LF este convertită într-un '\n' (LF). În Unix, sfârşitul de linie este reprezentat doar prin caracterul LF. În mod binar, atât în DOS-Windows cât şi în Unix, sfârşitul de linie este reprezentat doar prin caracterul LF.

Modul binar se utilizează de obicei pentru a scrie în fişier datele exact aşa cum sunt reprezentate în memorie (cu functiile fread, fwrite) - de exemplu pentru un număr intreg se va scrie reprezentarea internă a acestuia, pe 2 sau pe 4 octeti.

Modul text este utilizat mai ales pentru scrierea cu format (cu funcţiile fprintf, fscanf) - în cazul acesta pentru un număr întreg se vor scrie caracterele ASCII utilizate pentru a reprezenta cifrele acestuia (adică un şir de caractere cum ar fi “1” sau “542”).

Citire/scriere cu format

int fprintf(FILE \*fp, const char \*format, ...);

int fscanf(FILE \*fp, const char \*format, ...);

Funcţiile sunt utilizate pentru citire/scriere în mod text şi sunt asemănătoare cu printf/scanf (diferenţa fiind că trebuie dat pointerul la fişier ca prim parametru).

**Codul deplin al programului:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_STUDENT\_NUM 100

#define MAX\_NAME\_LEN 20

typedef struct inElev

{

char name[MAX\_NAME\_LEN];

int grades[5];

} inElev;

#define INPUT\_TMP "%s %d %d %d %d %d\n"

typedef struct outElev

{

char name[MAX\_NAME\_LEN];

float mean;

} outElev;

#define OUTPUT\_TMP "%s %f\n"

int studentNum = 0;

char repeated[MAX\_STUDENT\_NUM / 2][MAX\_STUDENT\_NUM];

void sort(outElev \*input)

{

outElev tmp;

for (int i = 0; i < studentNum; i++)

{

for (int j = i + 1; j < studentNum; j++)

{

if (input[i].mean < input[j].mean)

{

tmp = input[i];

input[i] = input[j];

input[j] = tmp;

}

}

}

}

int repeats(char \*name, outElev \*input)

{

int repeats = 0;

for (int i = 0, j = 0; i < studentNum; i++)

{

if (strcmp(name, input[i].name) == 0)

repeats++;

}

return (repeats > 1 ? 1 : 0);

}

int isInRepList(char \*name)

{

for (int i = 0, j = 0; i < studentNum; i++)

{

if (strcmp(name, repeated[i]) == 0)

return 1;

}

return 0;

}

void getRepeats(outElev \*input)

{

int j = 0;

for (int i = 0; i < studentNum; i++)

{

if (repeats(input[i].name, input) && !isInRepList(input[i].name))

{

strcpy(repeated[j], input[i].name);

j++;

}

}

}

int main()

{

// -- OPEN --

FILE \*in;

fopen\_s(&in, "input.txt", "r");

if (in == NULL)

{

printf("\nS-a produs o eroare in procesul deschiderii fisierului pentru citire.\n");

return 1;

}

FILE \*out;

fopen\_s(&out, "output.txt", "w");

if (out == NULL)

{

printf("\nS-a produs o eroare in procesul deschiderii fisierului pentru scriere.\n");

return 3;

}

FILE \*err;

fopen\_s(&err, "eronat.txt", "w");

if (err == NULL)

{

printf("\nS-a produs o eroare in procesul deschiderii fisierului pentru scriere.\n");

return 3;

}

int elements\_read = 0;

inElev input[MAX\_STUDENT\_NUM];

outElev output[MAX\_STUDENT\_NUM];

int elems\_wrote = 0;

while (!feof(in))

{

// -- READ --

elements\_read += fscanf(in, INPUT\_TMP,

input[studentNum].name,

&input[studentNum].grades[0],

&input[studentNum].grades[1],

&input[studentNum].grades[2],

&input[studentNum].grades[3],

&input[studentNum].grades[4]);

// -- CALCULATE --

strcpy(output[studentNum].name, input[studentNum].name);

int sum = 0;

for (int i = 0; i < 5; i++)

{

sum += input[studentNum].grades[i];

}

output[studentNum].mean = (float)sum / 5;

studentNum++;

}

sort(output);

getRepeats(output);

// -- WRITE --

for (int i = 0; i < studentNum; i++)

{

if (!isInRepList(output[i].name))

elems\_wrote += fprintf(out, OUTPUT\_TMP, output[i].name, output[i].mean);

if (isInRepList(input[i].name))

elems\_wrote += fprintf(err, INPUT\_TMP,

input[i].name,

input[i].grades[0],

input[i].grades[1],

input[i].grades[2],

input[i].grades[3],

input[i].grades[4]);

}

// -- CLOSE --

fclose(in);

if (elements\_read < studentNum \* 6)

{

printf("\nS-a produs o eroare in procesul de citire a datelor.\n");

return 2;

}

fclose(out);

fclose(err);

if (elems\_wrote <= 0)

{

printf("\nS-a produs o eroare in procesul de scriere a datelor.\n");

return 4;

}

return 0;

}

**Execuția programului:**









**Concluzii:**

* Stocarea informației în fișiere text este foarte utilă pentru sarcini ce necesită citirea datelor de către un om.
* Deși ele nu sunt atât de eficiente ca fișierele binare, ele oferă comprehensibilitate pentru oricine, nu doar pentru programatori.
* De asemenea, fișierele text permit compatibilitatea între sisteme de operare, deoarece ele dețin același format pe o mașină de 32 de biți, una de 64 de biți, una care lucrează pe Linux, una care lucrează pe Windows ș.a.m.d. spre deosebire de fișierele binare, care sunt mai dependente de mărimea fiecărui tip de date stocat în ele.

**Surse:**

* ocw.cs.pub.ro, Cursul de programare in C, Operaţii cu fişiere. Aplicaţii folosind fişiere.

<https://ocw.cs.pub.ro/courses/programare/laboratoare/lab12>