

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**gr. IA-233, Chistol Maxim**

**Raport**

**pentru lucrarea de laborator Nr.3**

***la cursul de “Structuri de Date si Algoritmi”***

Verificat:

**Maria Guțu,** *Lector Universitar.*

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău 2024**

**Cuprins:**

[Scopul lucrării: 3](#_Toc160012318)

[Sarcina 4](#_Toc160012319)

[Concluzie: 4](#_Toc160012320)

**Scopul lucrării:**

Programarea algoritmilor de prelucrare a structurilor și a tablourilor de la structuri prin utilizarea funcțiilor, pointerilor, alocării dinamice a memoriei în limbajul C.

# **Sarcina**

Pentru tabloul unidimensional dat cu elemente de tip structură (conform variantelor) să se afișeze la ecran următorul meniu recursiv de opțiuni:

1. Introducerea elementelor tabloului de la tastatură.

2. Afișarea elementelor tabloului la ecran.

3. Sortarea tabloului crescător/descrescător (2 tehnici de sortare la alegere).

0. Ieșire din program.

**Sarcina conform variantei:**

4. Structura Cinematograf cu câmpurile: denumirea, adresa, telefonul, numărul de locuri, prețul biletelor.

# **Codul:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Cinematograf {

char denumire[50];

char adresa[100];

char telefon[15];

int numarLocuri;

float pretBilete;

};

void introducereDate(struct Cinematograf cinema[], int n) {

for (int i = 0; i < n; ++i) {

printf("\nCinematograful #%d:\n", i + 1);

printf("Denumirea: ");

scanf("%s", cinema[i].denumire);

printf("Adresa: ");

scanf("%s", cinema[i].adresa);

printf("Telefon: ");

scanf("%s", cinema[i].telefon);

printf("Numar de locuri: ");

scanf("%d", &cinema[i].numarLocuri);

printf("Pretul biletelor: ");

scanf("%f", &cinema[i].pretBilete);

}

}

void afisareDate(const struct Cinematograf cinema[], int n) {

printf("\n%-20s%-20s%-15s%-15s%-15s\n", "Denumire", "Adresa", "Telefon", "Locuri", "Pret Bilete");

for (int i = 0; i < n; ++i) {

printf("%-20s%-20s%-15s%-15d%-15.2f\n", cinema[i].denumire, cinema[i].adresa, cinema[i].telefon, cinema[i].numarLocuri, cinema[i].pretBilete);

}

}

void countingSort(struct Cinematograf cinema[], int n) {

// Găsește valoarea maximă din câmpul 'numarLocuri'

int maxLocuri = cinema[0].numarLocuri;

for (int i = 1; i < n; ++i) {

if (cinema[i].numarLocuri > maxLocuri) {

maxLocuri = cinema[i].numarLocuri;

}

}

// Inițializare vector de frecvență și vector sortat

int \*frecventa = (int \*)malloc((maxLocuri + 1) \* sizeof(int));

struct Cinematograf \*sortat = (struct Cinematograf \*)malloc(n \* sizeof(struct Cinematograf));

// Initializează vectorul de frecvență cu 0

memset(frecventa, 0, (maxLocuri + 1) \* sizeof(int));

// Calculează frecvența fiecărui număr de locuri

for (int i = 0; i < n; ++i) {

frecventa[cinema[i].numarLocuri]++;

}

// Calculează pozițiile finale ale fiecărui element

for (int i = 1; i <= maxLocuri; ++i) {

frecventa[i] += frecventa[i - 1];

}

// Construiește vectorul sortat

for (int i = n - 1; i >= 0; --i) {

sortat[frecventa[cinema[i].numarLocuri] - 1] = cinema[i];

frecventa[cinema[i].numarLocuri]--;

}

// Copiază vectorul sortat în vectorul original

for (int i = 0; i < n; ++i) {

cinema[i] = sortat[i];

}

// Eliberează memoria alocată dinamic

free(frecventa);

free(sortat);

printf("\nSortare realizata cu Counting Sort.\n");

}

int gasesteLungimeMaxima(struct Cinematograf cinema[], int n) {

int maxLungime = strlen(cinema[0].denumire);

for (int i = 1; i < n; ++i) {

int lungimeCurenta = strlen(cinema[i].denumire);

if (lungimeCurenta > maxLungime) {

maxLungime = lungimeCurenta;

}

}

return maxLungime;

}

void radixSort(struct Cinematograf cinema[], int n) {

// Găsește lungimea maximă a șirurilor

int maxLungime = gasesteLungimeMaxima(cinema, n);

// Aplică sortare pe fiecare caracter, de la cel mai semnificativ către cel mai puțin semnificativ

for (int i = maxLungime - 1; i >= 0; --i) {

// Initializează vector de coșuri (buckets)

struct Cinematograf coșuri[256][n];

int contorCoșuri[256] = {0};

// Distribuie elementele în coșuri

for (int j = 0; j < n; ++j) {

int index = (unsigned char)cinema[j].denumire[i];

memcpy(coșuri[index] + contorCoșuri[index], &cinema[j], sizeof(struct Cinematograf));

contorCoșuri[index]++;

}

// Colectează elementele din coșuri

int indexCinema = 0;

for (int j = 0; j < 256; ++j) {

for (int k = 0; k < contorCoșuri[j]; ++k) {

memcpy(&cinema[indexCinema], &coșuri[j][k], sizeof(struct Cinematograf));

indexCinema++;

}

}

}

printf("\nSortare realizata cu Radix Sort.\n");

}

int main() {

const int MAX\_CINEME = 10;

struct Cinematograf cinema[MAX\_CINEME];

int n = 0;

int optiune;

do {

printf("\n----- Meniu -----\n");

printf("1. Introducere date cinematografe\n");

printf("2. Afisare date cinematografe\n");

printf("3. Counting Sort\n");

printf("4. Radix Sort\n");

printf("0. Iesire\n");

printf("Alege optiunea: ");

scanf("%d", &optiune);

switch (optiune) {

case 1:

printf("Introduceti numarul de cinematografe (maxim %d): ", MAX\_CINEME);

scanf("%d", &n);

introducereDate(cinema, n);

break;

case 2:

afisareDate(cinema, n);

break;

case 3:

countingSort(cinema, n);

break;

case 4:

radixSort(cinema, n);

break;

case 0:

printf("Programul se inchide.\n");

break;

default:

printf("Optiune invalida! Alege din nou.\n");

break;

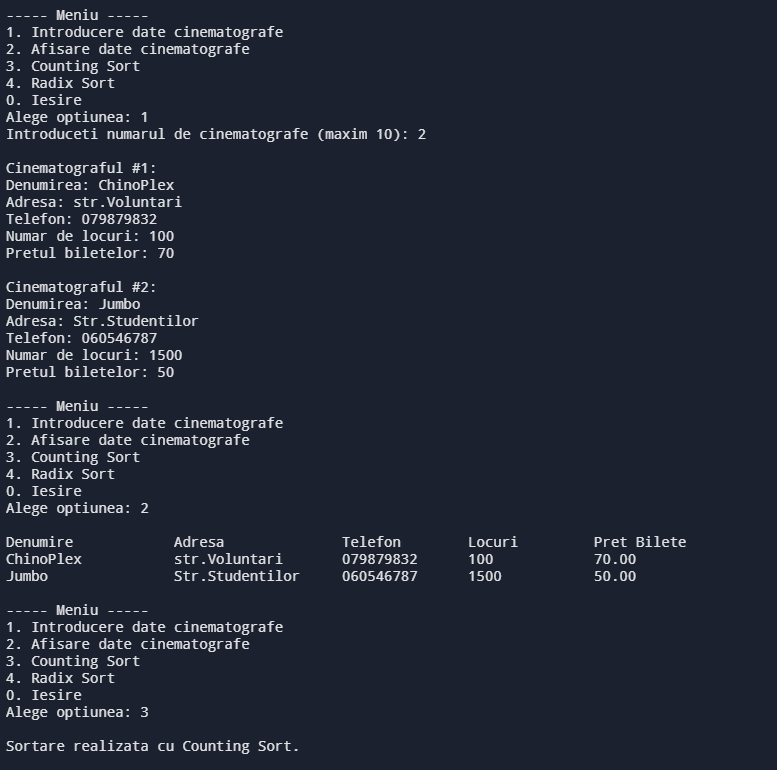
}

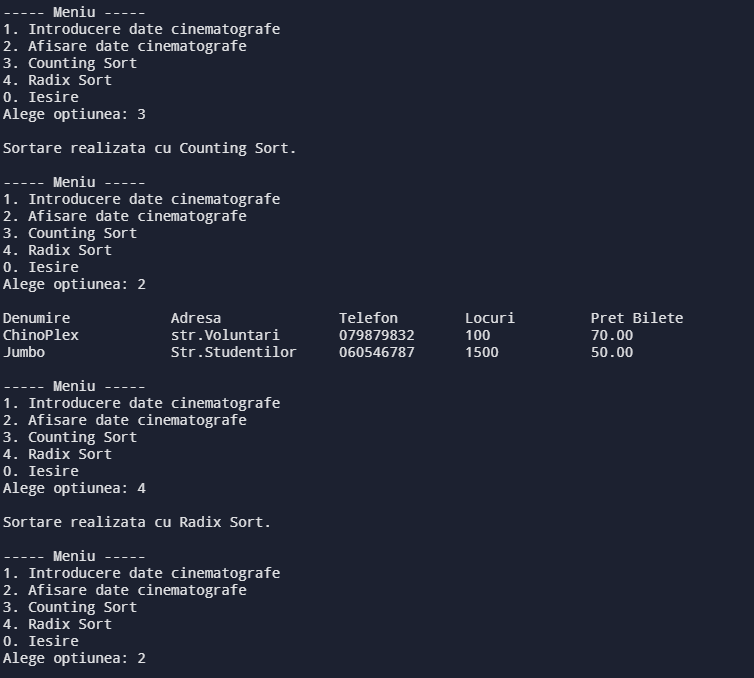
} while (optiune != 0);

return 0;

}

# **Rezultatele:**





# **Concluzie:**

În concluzie, programul realizat în limbajul C a ilustrat eficiența utilizării funcțiilor, pointerilor și alocării dinamice a memoriei în gestionarea datelor cinematografelor. Prin definirea și manipularea unei structuri de date, am permis utilizatorului să introducă și să afișeze informații despre cinematografe. De asemenea, funcționalitățile de sortare crescătoare și descrescătoare, implementate prin algoritmii Counting Sort și Radix Sort, au adăugat un nivel suplimentar de complexitate programului.

Utilizarea pointerilor în alocarea dinamică a memoriei ne-a oferit flexibilitate în manipularea datelor și eficiență în gestionarea resurselor. Astfel, am putut adapta dimensiunea tabloului de structuri în timpul execuției, fără a depinde de o dimensiune fixă predefinită.

Prin această implementare, am subliniat beneficiile utilizării limbajului C pentru gestionarea eficientă a memoriei și manipularea flexibilă a datelor. În ansamblu, programul oferă o platformă simplă și eficientă pentru prelucrarea și organizarea informațiilor legate de cinematografe.