****

**MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII**

**AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**Calancea Cătălin**

**MI-222**

**Raport**

**pentru lucrarea de laborator Nr.7**

***la cursul de “Structuri de date şi algoritmi”***

Verificat:

**Guțu Maria,** *doctor, conf. univ.*

Departamentul Informatică şi IS,

Facultatea FCIM, UTM

**Chișinău – 2023**

**Scopul lucrării:** Obținerea deprinderilor practice de implementare și de utilizare a tipului abstract de date „Stack” și „Queue” în limbajul C cu asigurarea operațiilor de prelucrare de bază.

**Nr. variantei:** 4

**Condiția problemei ( sarcinii de lucru ) :**

Să se scrie un program în limbajul C care contine Structura Imobil cu câmpurile: proprietarul, tipul, adresa, suprafața, costul.Funcția mаin() va afișa la ecran următorul meniu de opțiuni de bază:

1. Crearea unei stive dinamice

2. Citirea datelor referitoare la elementele stivei de la tastatură.

3. Afișarea datelor stivei într-un fișier stiva.txt/stiva.bin.

4. Căutarea elementului maximal din stivă după un câmp numeric.

5. Copierea datelor din stivă într-un șir în așteptare (queue).

6. Afișarea datelor șirului în așteptare într-un fișier queue.txt/queue.bin.

7. Determinarea lungimii stivei (numărul de elemente).

8. Modificarea câmpurilor unui element din stivă.

9. Afișarea(Adăugarea) datelor modificate a elementului din stivă la sfârșitul fișierului stiva.txt/stiva.bin.

10. Eliberarea memoriei alocate pentru stivă.

11. Eliberarea memoriei alocate pentru Queue.

0. Ieșire din program.

**Codul programului:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

struct Imobil {

char proprietarul[50];

char tipul[50];

char adresa[100];

float suprafata;

float costul;

};

void creareStiva(struct Imobil\*\* stiva, int\* numarElemente) {

printf("Introduceti numarul de elemente din stiva: ");

scanf("%d", numarElemente);

getchar(); // pentru a consuma caracterul newline ('\n') lăsat în buffer de scanf

\*stiva = (struct Imobil\*)malloc(\*numarElemente \* sizeof(struct Imobil));

if (\*stiva == NULL) {

printf("Alocarea memoriei a esuat.\n");

return;

}

printf("Stiva a fost creata cu succes.\n");

}

void citireDate(struct Imobil\* stiva, int numarElemente) {

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

printf("Introduceti datele pentru imobilul %d:\n", i + 1);

printf("Proprietarul: ");

fgets(stiva[i].proprietarul, 50, stdin);

stiva[i].proprietarul[strcspn(stiva[i].proprietarul, "\n")] = '\0';

printf("Tipul: ");

fgets(stiva[i].tipul, 50, stdin);

stiva[i].tipul[strcspn(stiva[i].tipul, "\n")] = '\0';

printf("Adresa: ");

fgets(stiva[i].adresa, 100, stdin);

stiva[i].adresa[strcspn(stiva[i].adresa, "\n")] = '\0';

printf("Suprafata: ");

scanf("%f", &stiva[i].suprafata);

getchar();

printf("Costul: ");

scanf("%f", &stiva[i].costul);

getchar();

printf("\n");

}

}

void afisareStiva(struct Imobil\* stiva, int numarElemente, const char\* fisier) {

FILE\* file = fopen(fisier, "w");

if (file == NULL) {

printf("Nu s-a putut deschide fisierul.\n");

return;

}

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

fprintf(file, "Imobilul %d:\n", i + 1);

fprintf(file, "Proprietarul: %s\n", stiva[i].proprietarul);

fprintf(file, "Tipul: %s\n", stiva[i].tipul);

fprintf(file, "Adresa: %s\n", stiva[i].adresa);

fprintf(file, "Suprafata: %.2f\n", stiva[i].suprafata);

fprintf(file, "Costul: %.2f\n", stiva[i].costul);

fprintf(file, "\n");

}

fclose(file);

printf("Datele stivei au fost salvate in fisierul %s.\n", fisier);

}

void cautareMaximal(struct Imobil\* stiva, int numarElemente) {

float maxCost = -1.0;

int pozitieMax = 0;

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

if (stiva[i].costul > maxCost) {

maxCost = stiva[i].costul;

pozitieMax = i;

}

}

printf("Elementul cu costul maxim:\n");

printf("Proprietarul: %s\n", stiva[pozitieMax].proprietarul);

printf("Tipul: %s\n", stiva[pozitieMax].tipul);

printf("Adresa: %s\n", stiva[pozitieMax].adresa);

printf("Suprafata: %.2f\n", stiva[pozitieMax].suprafata);

printf("Costul: %.2f\n", stiva[pozitieMax].costul);

}

void copiereStivaInQueue(struct Imobil\* stiva, int numarElemente, struct Imobil\* queue, int\* rear) {

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

strcpy(queue[\*rear].proprietarul, stiva[i].proprietarul);

strcpy(queue[\*rear].tipul, stiva[i].tipul);

strcpy(queue[\*rear].adresa, stiva[i].adresa);

queue[\*rear].suprafata = stiva[i].suprafata;

queue[\*rear].costul = stiva[i].costul;

(\*rear)++;

}

printf("Datele au fost copiate din stiva in queue.\n");

}

void afisareQueue(struct Imobil\* queue, int rear, const char\* fisier) {

FILE\* file = fopen(fisier, "w");

if (file == NULL) {

printf("Nu s-a putut deschide fisierul.\n");

return;

}

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

fprintf(file, "Imobilul %d:\n", i + 1);

fprintf(file, "Proprietarul: %s\n", queue[i].proprietarul);

fprintf(file, "Tipul: %s\n", queue[i].tipul);

fprintf(file, "Adresa: %s\n", queue[i].adresa);

fprintf(file, "Suprafata: %.2f\n", queue[i].suprafata);

fprintf(file, "Costul: %.2f\n", queue[i].costul);

fprintf(file, "\n");

}

fclose(file);

printf("Datele queue-ului au fost salvate in fisierul %s.\n", fisier);

}

int determinaLungimeStiva(int numarElemente) {

return numarElemente;

}

void modificareElement(struct Imobil\* stiva, int numarElemente) {

int pozitie;printf("Introduceti pozitia elementului pe care doriti sa il modificati: ");

scanf("%d", &pozitie);

getchar();

if (pozitie < 1 || pozitie > numarElemente) {

printf("Pozitie invalida.\n");

return;

}

struct Imobil\* element = &stiva[pozitie - 1];

printf("Introduceti noile date pentru elementul %d:\n", pozitie);

printf("Proprietarul: ");

fgets(element->proprietarul, 50, stdin);

element->proprietarul[strcspn(element->proprietarul, "\n")] = '\0';printf("Tipul: ");

fgets(element->tipul, 50, stdin);

element->tipul[strcspn(element->tipul, "\n")] = '\0';

printf("Adresa: ");

fgets(element->adresa, 100, stdin);

element->adresa[strcspn(element->adresa, "\n")] = '\0';

printf("Suprafata: ");

scanf("%f", &element->suprafata);

getchar();

printf("Costul: ");

scanf("%f", &element->costul);

getchar();

printf("Elementul %d a fost modificat cu succes.\n", pozitie);

}

void adaugareLaSfarsit(struct Imobil\* stiva, int numarElemente, const char\* fisier) {

FILE\* file = fopen(fisier, "a");

if (file == NULL) {

printf("Nu s-a putut deschide fisierul.\n");

return;

}struct Imobil\* element = &stiva[numarElemente - 1];

fprintf(file, "Element adaugat la sfarsit:\n");

fprintf(file, "Proprietarul: %s\n", element->proprietarul);

fprintf(file, "Tipul: %s\n", element->tipul);

fprintf(file, "Adresa: %s\n", element->adresa);

fprintf(file, "Suprafata: %.2f\n", element->suprafata);

fprintf(file, "Costul: %.2f\n", element->costul);

fprintf(file, "\n");

fclose(file);

printf("Datele elementului adaugat la sfarsit au fost salvate in fisierul %s.\n", fisier);

}

void eliberareMemorieStiva(struct Imobil\* stiva) {

free(stiva);

printf("Memoria alocata pentru stiva a fost eliberata.\n");

}

void eliberareMemorieQueue(struct Imobil\* queue) {

free(queue);

printf("Memoria alocata pentru queue a fost eliberata.\n");

}

int main() {

struct Imobil\* stiva = NULL;

int numarElemente = 0;struct Imobil\* queue = NULL;

int rear = 0;

int option;

do {

printf("Meniu de optiuni:\n");

printf("1. Crearea unei stive dinamice\n");

printf("2. Citirea datelor referitoare la elementele stivei de la tastatura\n");

printf("3. Afisarea datelor stivei intr-un fisier stiva.txt/stiva.bin\n");

printf("4. Cautarea elementului maximal din stiva dupa un camp numeric\n");

printf("5. Copierea datelor din stiva intr-un sir in asteptare (queue)\n");

printf("6. Afisarea datelor sirului in asteptare intr-un fisier queue.txt/queue.bin\n");

printf("7. Determinarea lungimii stivei (numarul de elemente)\n");

printf("8. Modificarea campurilor unui element din stiva\n");

printf("9. Afisarea (Adaugarea) datelor modificate a elementului din stiva la sfarsitul fisierului stiva.txt/stiva.bin\n");

printf("10. Eliberarea memoriei alocate pentru stiva.\n");

printf("11. Eliberarea memoriei alocate pentru Queue.\n");

printf("0. Iesire din program.\n");

printf("Introduceti optiunea dorita: ");

scanf("%d", &option);

getchar();

switch (option) {

case 1:

creareStiva(&stiva, &numarElemente);

break;

case 2:

citireDate(stiva, numarElemente);

break;

case 3:

afisareStiva(stiva, numarElemente, "stiva.txt");

break;

case 4:

cautareMaximal(stiva, numarElemente);

break;

case 5:

queue = (struct Imobil\*)malloc(numarElemente \* sizeof(struct Imobil));

if (queue == NULL) {

printf("Alocarea memoriei pentru queue a esuat.\n");

break;

}

copiereStivaInQueue(stiva, numarElemente, queue, &rear);

break;

case 6:

afisareQueue(queue, rear, "queue.txt");

break;

case 7:

printf("Lungimea stivei este: %d\n", determinaLungimeStiva(numarElemente));

break;

case 8:

modificareElement(stiva, numarElemente);

break;

case 9:

adaugareLaSfarsit(stiva, numarElemente, "stiva.txt");

break;

case 10:

eliberareMemorieStiva(stiva);

stiva = NULL;

numarElemente = 0;

break;

case 11:

eliberareMemorieQueue(queue);

queue = NULL;

rear = 0;

break;

case 0:

printf("Programul s-a terminat.\n");

break;

default:

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

break;

}

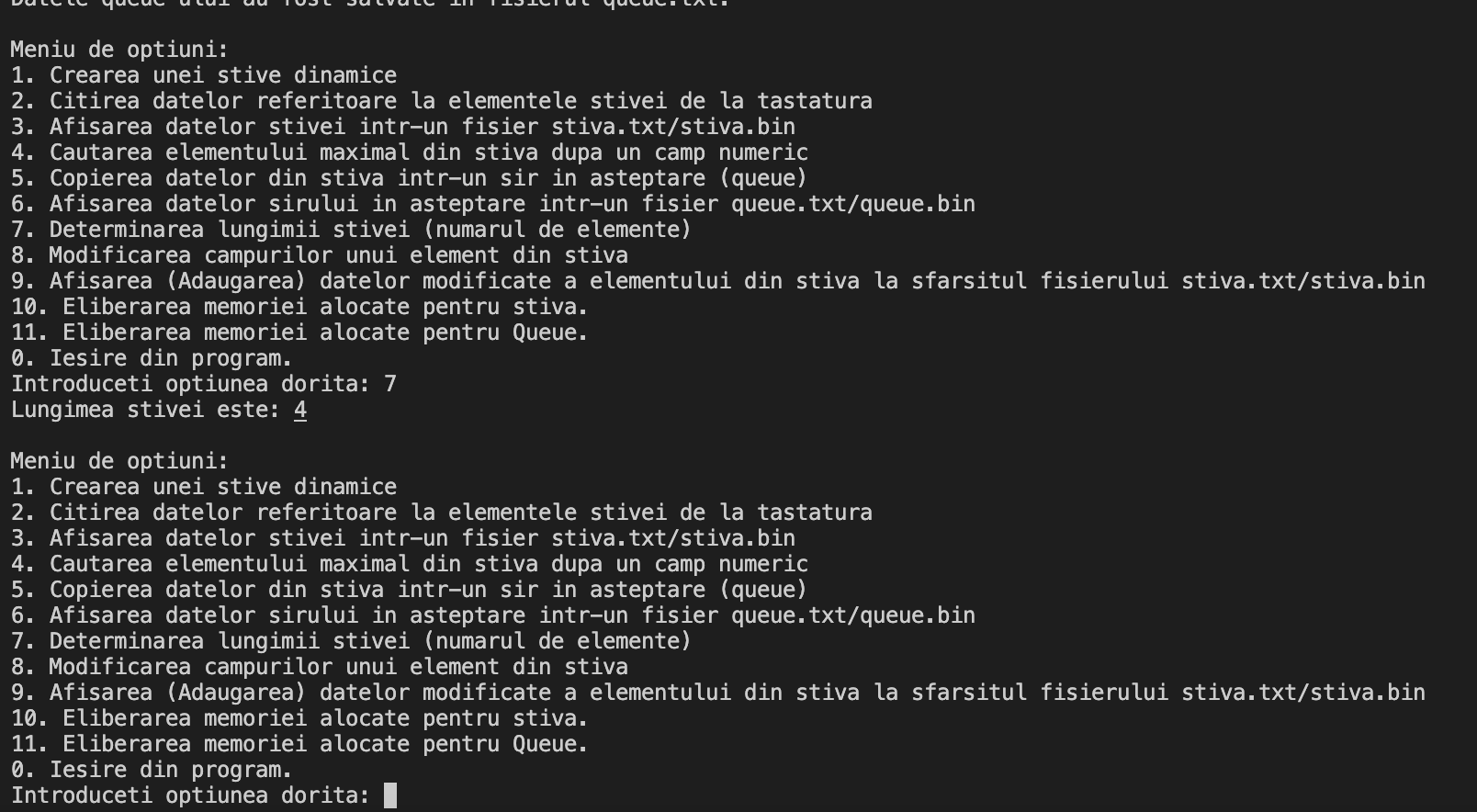
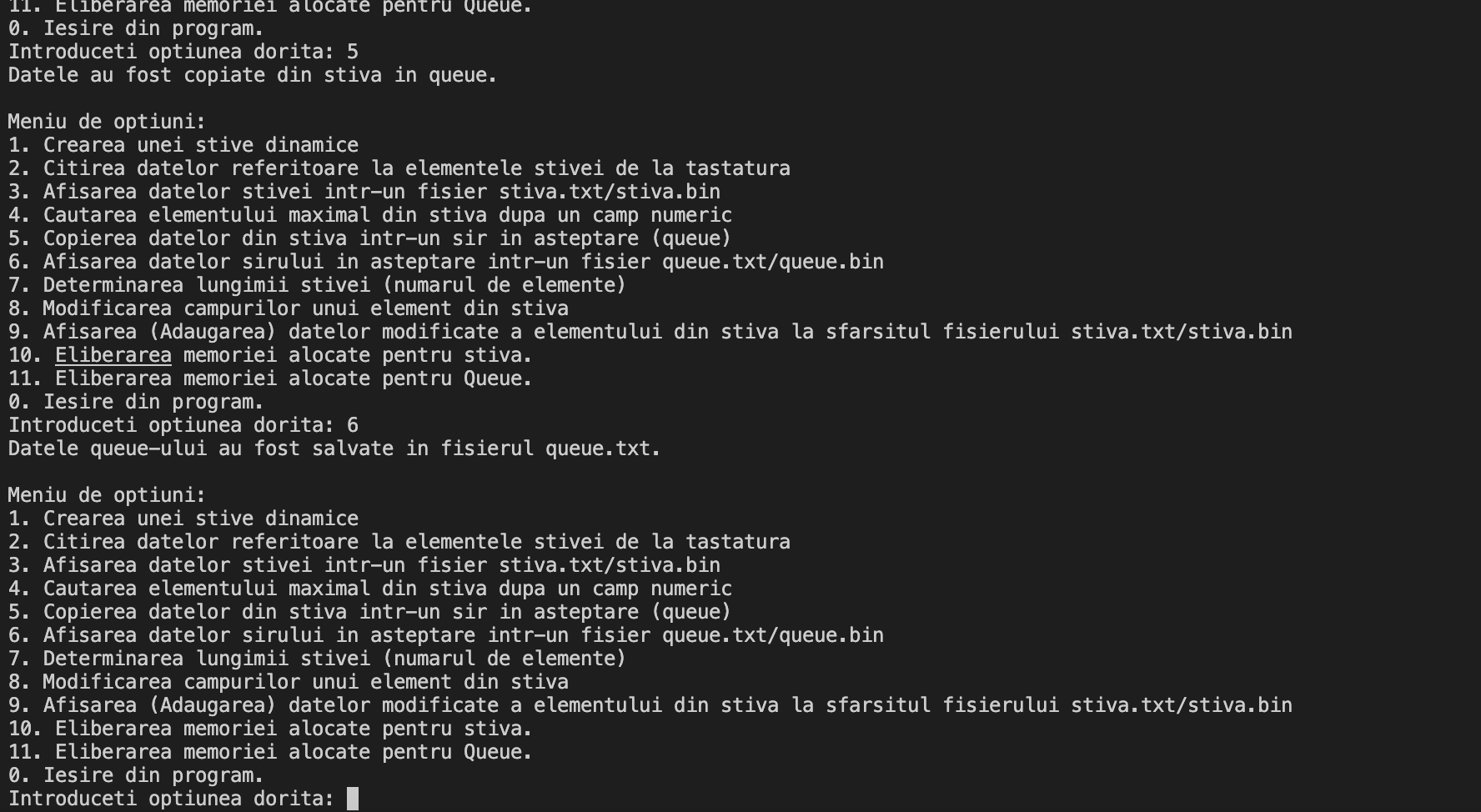
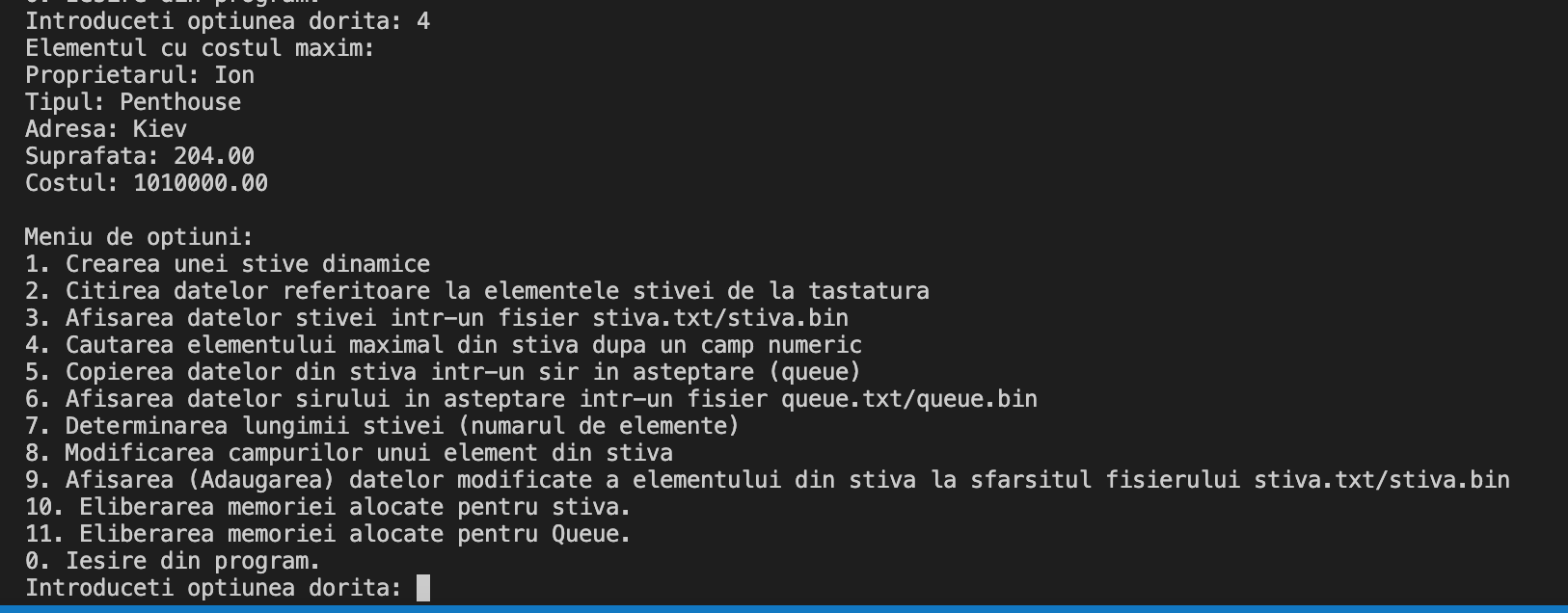
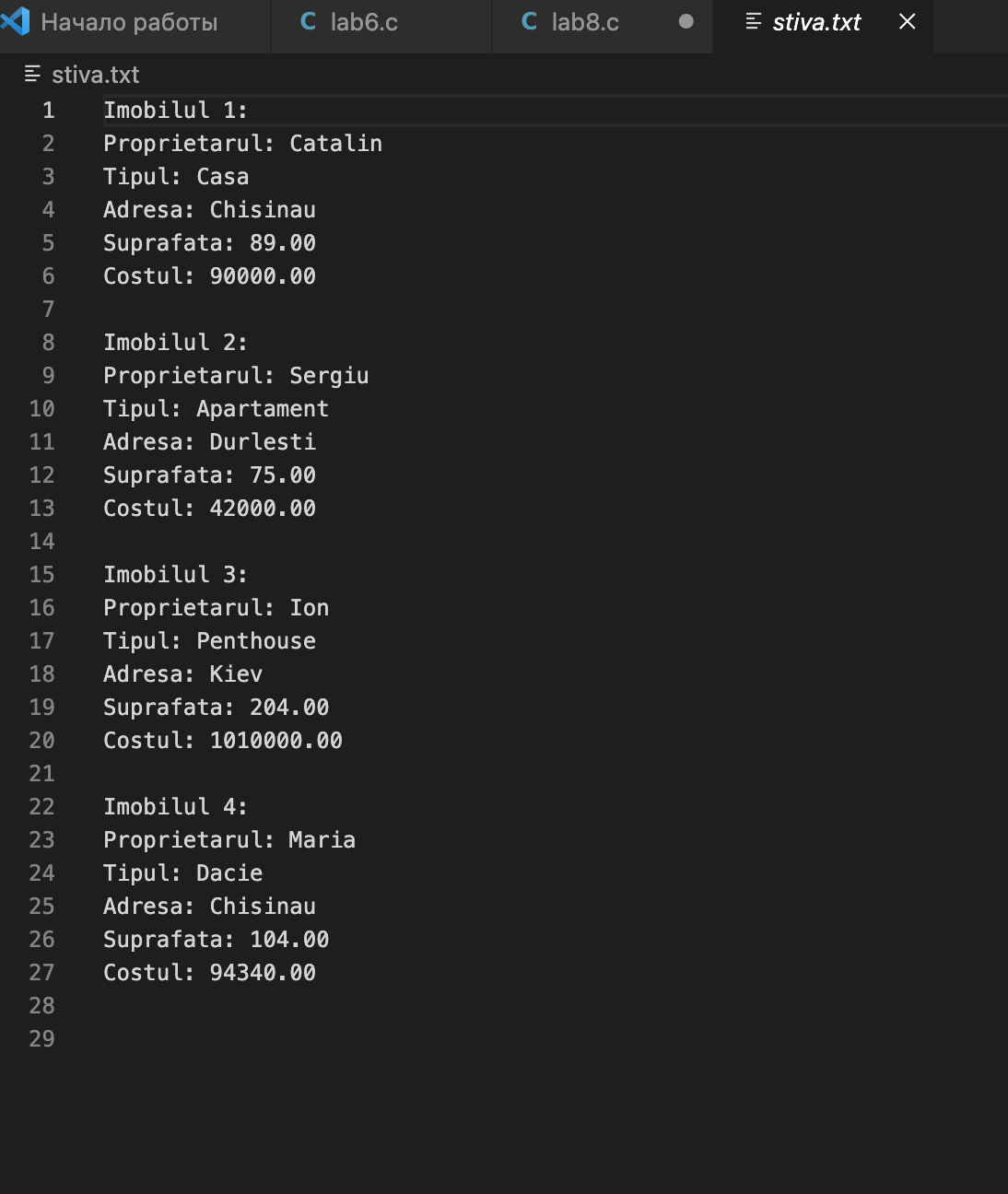
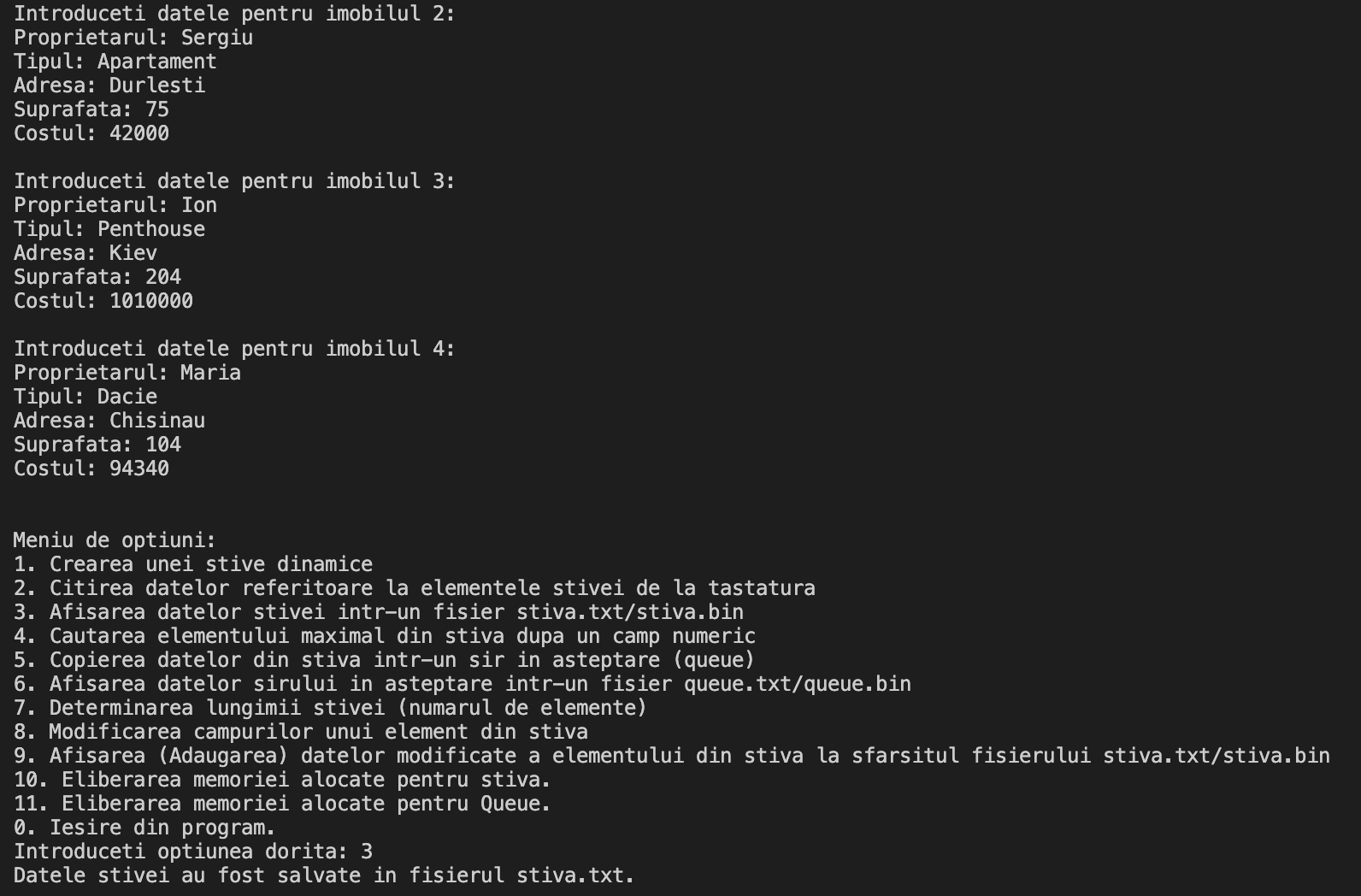
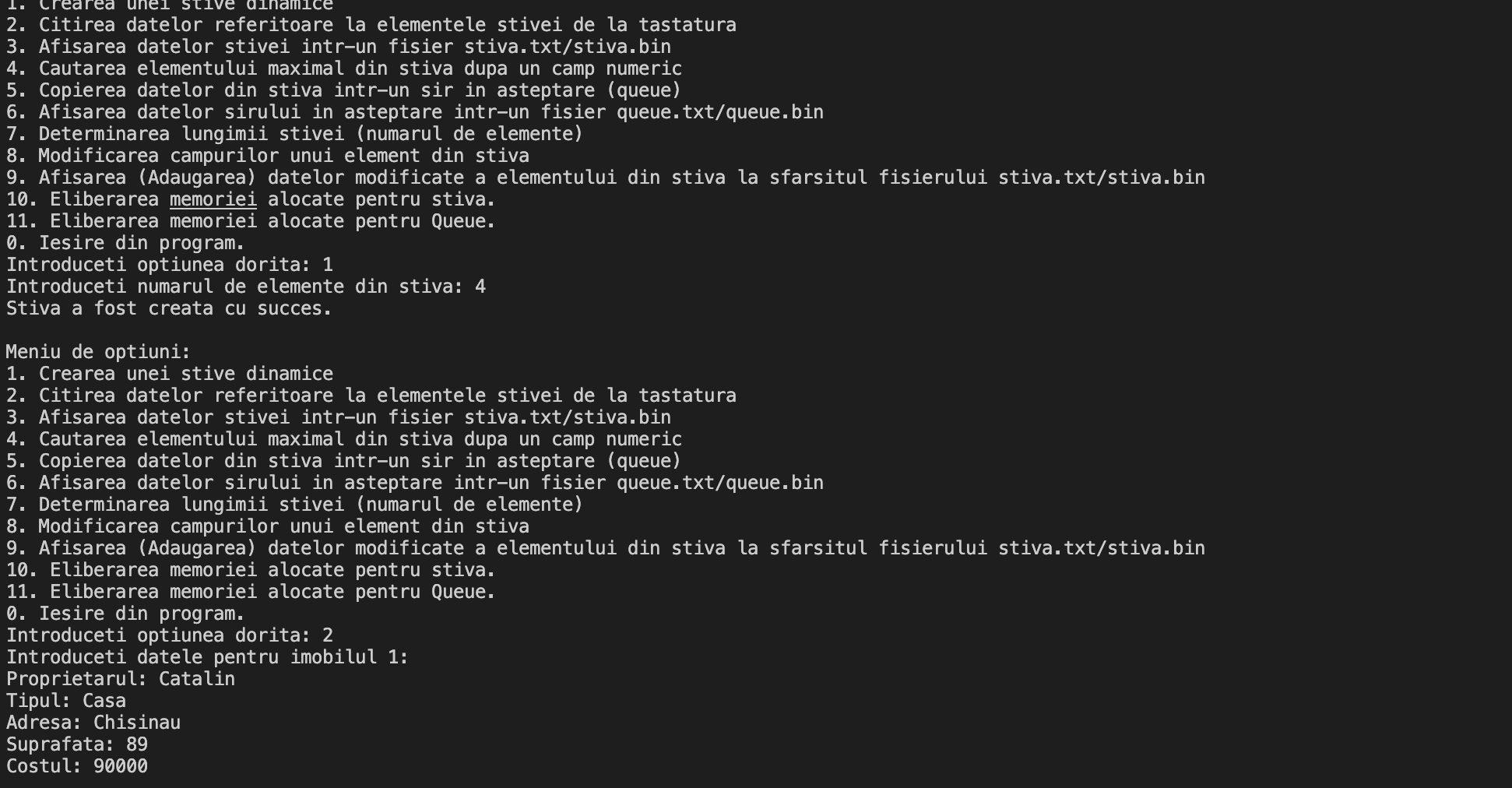
printf("\n");

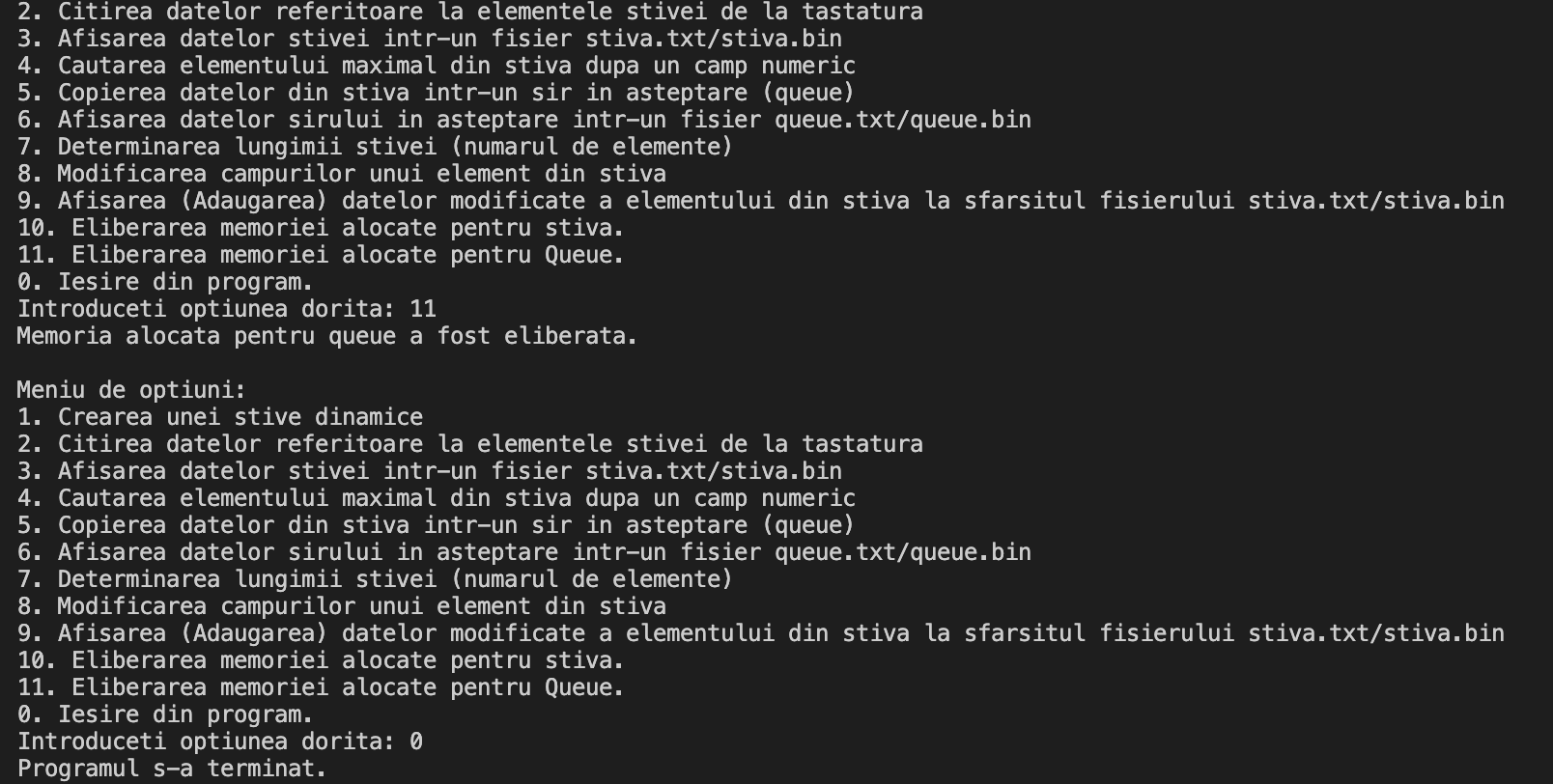
} while (option != 0);

return 0;

}

**Output:**

****

****

**În concluzie,** am învățat despre tipurile abstracte de date "Stack" și "Queue" și am dobândit deprinderi practice de implementare și utilizare a acestora în limbajul C. Am înțeles că un "Stack" reprezintă o structură de date în care elementele sunt adăugate sau eliminate doar la un capăt, numit vârf, urmând principiul LIFO (Last-In, First-Out), în timp ce un "Queue" reprezintă o structură de date în care elementele sunt adăugate la un capăt, numit coadă, și eliminate de la celălalt capăt, numit față, urmând principiul FIFO (First-In, First-Out).Pentru implementarea unui "Stack", am folosit un vector sau o listă simplu înlănțuită, iar operațiile de bază au inclus adăugarea unui element în vârf (push), eliminarea unui element din vârf (pop) și obținerea valorii elementului din vârf (top). Am învățat cum să gestionăm corect indicii și să ne asigurăm că nu depășim limitele vectorului sau listei.Pentru implementarea unui "Queue", am folosit, de asemenea, un vector sau o listă simplu înlănțuită, iar operațiile de bază au inclus adăugarea unui element la sfârșitul cozii (enqueue), eliminarea unui element de la începutul cozii (dequeue) și obținerea valorii primului element din coadă (front). Am învățat cum să manipulăm indicele de început și cel de sfârșit al cozii, asigurându-ne că aceștia se actualizează corect în funcție de operațiile efectuate. Implementarea și utilizarea eficientă a acestor tipuri abstracte de date ne oferă unelte puternice pentru rezolvarea unei varietăți de probleme. Am învățat că "Stack"-urile sunt utile în situațiile în care avem nevoie să stocăm și să accesezăm elemente în ordine inversă, iar "Queue"-urile sunt utile în situațiile în care avem nevoie să menținem ordinea originală a elementelor.În final, deprinderile practice dobândite în implementarea și utilizarea "Stack"-urilor și "Queue"-urilor în limbajul C ne permit să rezolvăm diverse probleme de prelucrare a datelor într-un mod eficient și optim.