Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

**RAPORT**

Lucrare de laborator Nr.7

*la Programarea Calculatoarelor*

Tema: Structuri și tablouri de la structuri

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

A verificat: lect. asist. Mititelu Vitalii

Chişinău 2021

**Scopul:** 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 de opțiuni:

1. Alocarea dinamică a memoriei pentru tabloul de structuri.

2. Introducerea elementelor tabloului de la tastatură.

3. Afișarea elementelor tabloului la ecran.

4. Adăugarea unui element nou la sfârșit.

5. Modificarea elementului tabloului.

6. Căutarea elementului tabloului.

7. Sortarea tabloului.

8. Eliminarea elementului indicat din tablou.

9. Eliberarea memoriei alocate pentru tablou.

0. Ieșire din program.

Să se elaboreze funcțiile pentru implementarea opțiunilor meniului.

**Varianta 15:**

Structura Computer cu câmpurile: modelul, procesorul, memoria, viteza, prețul.

**Rezumat:**

**Structuri de date**

O *structură de date* este o colecție de informațiii grupate sub un singur nume. Aceste informații, cunoscute drept *membri*, pot avea tipuri de date diferite și lungimi diferite. Structurile de date pot fi definite în C++ folosind următoarea sintaxă:  
  
struct nume\_tip {  
tip\_data\_1 id\_membru\_1;  
tip\_data\_2 id\_membru\_2;  
tip\_data\_3 id\_membru\_3;  
.  
.  
} denumiri\_obiecte;  
  
unde nume\_tip este o denumire pentru tipul structură, denumiri\_obiecte poate fi un set de identificatori valizi de obiecte care sunt de acel tip structură. Între acolade {} avem o listă cu informațiile membre, fiecare dintre ele fiind precizată cu tip de dată și un identificator.  
De exemplu:

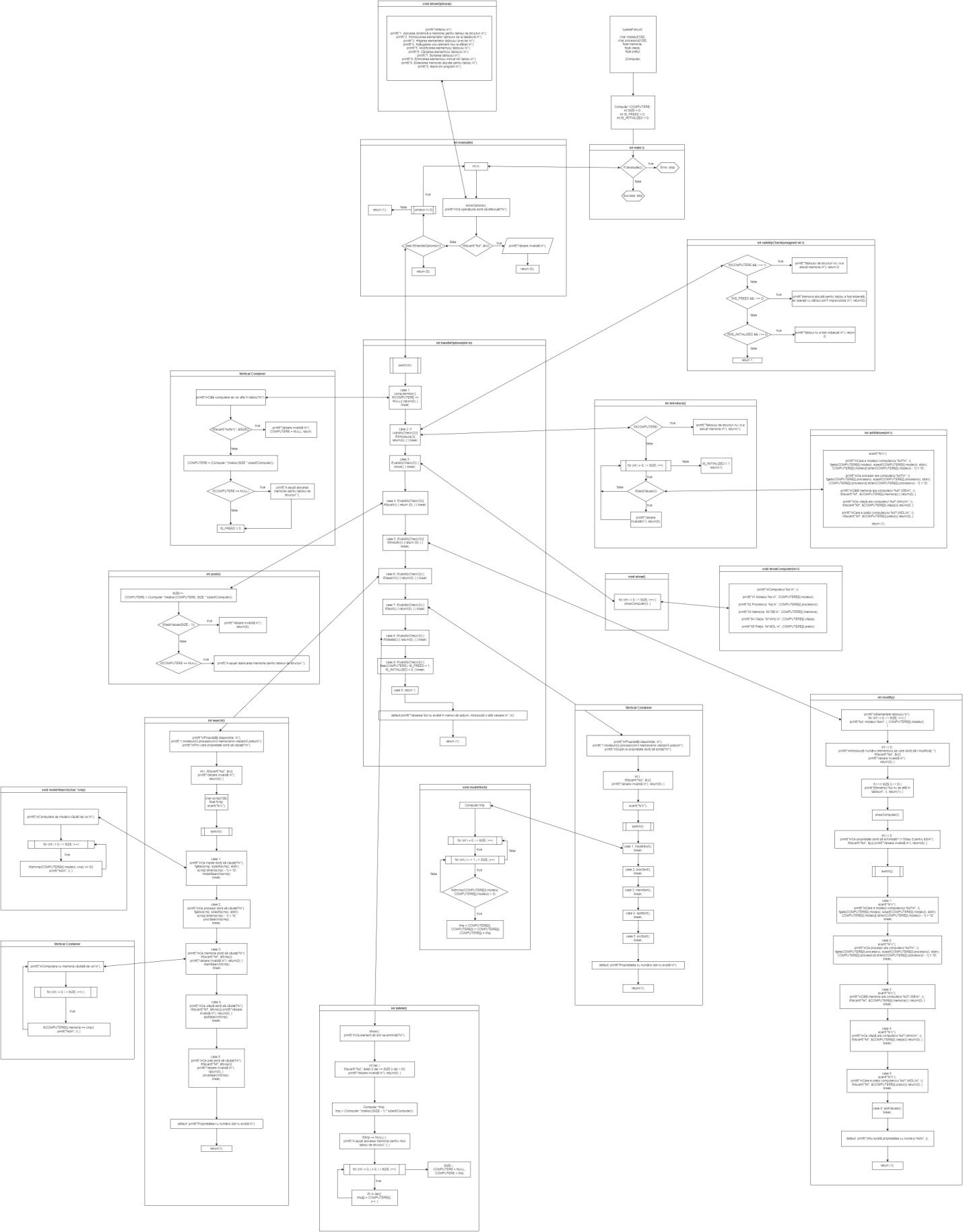
|  |  |  |
| --- | --- | --- |
| 1 2 3 4 5 6 7 | struct produs {  int greutate;  double pret;  } ;  produs mar;  produs banana, pepene; |  |

Avem o definiție de tip structură, denumit produs, care are doi membri: greutate și pret, fiecare de un alt tip de dată fundamental. Această definiție creează un nou tip (produs), care este apoi folosit pentru a declara trei obiecte (variabile) de acest tip: mar, banana și pepene. Să observăm că, odată definit, tipul produs poate fi folosit la fel ca orice alt tip de dată fundamental.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37  38 | // tablou de structuri  #include <iostream>  #include <string>  #include <sstream>  using namespace std;  struct filme\_t {  string titlu;  int an;  } filme [3];  void afis\_film (filme\_t film);  int main ()  {  string mystr;  int n;  for (n=0; n<3; n++)  {  cout << "Tasteaza titlul: ";  getline (cin,filme[n].titlu);  cout << "Tasteaza anul: ";  getline (cin,mystr);  stringstream(mystr) >> filme[n].an;  }  cout << "\nAi introdus urmatoarele filme:\n";  for (n=0; n<3; n++)  afis\_film (filme[n]);  return 0;  }  void afis\_film (filme\_t film)  {  cout << film.titlu;  cout << " (" << film.an << ")\n";  } |

Deoarece structurile sunt tipuri de date, ele pot fi folosite ca tip de bază pentru tablouri atunci când construim tabele sau baze de date cu ele

**Schema bloc a algoritmului de soluționare a problemei:**



**Codul deplin al programului:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct{

    char modelul[100];

    char procesorul[100];

    float memoria;

    float viteza;

    float pretul;

}Computer;

Computer \*COMPUTERE;

int SIZE = 0;

int IS\_FREED = 0;

int IS\_INITIALIZED = 0;

void showOptions() {

    printf("\nMeniu:\n");

    printf("1. Alocarea dinamică a memoriei pentru tabloul de structuri.\n");

    printf("2. Introducerea elementelor tabloului de la tastatură.\n");

    printf("3. Afișarea elementelor tabloului la ecran.\n");

    printf("4. Adăugarea unui element nou la sfârșit.\n");

    printf("5. Modificarea elementului tabloului.\n");

    printf("6. Căutarea elementului tabloului.\n");

    printf("7. Sortarea tabloului.\n");

    printf("8. Eliminarea elementului indicat din tablou.\n");

    printf("9. Eliberarea memoriei alocate pentru tablou.\n");

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

}

void computerAlloc(){

    printf("\nCâte computere se vor afla în tablou?\n");

    if(!scanf("%d%\*c", &SIZE)){

        printf("Valoare invalidă.\n");

        COMPUTERE = NULL;

        return;

    }

    COMPUTERE = (Computer \*)malloc(SIZE \* sizeof(Computer));

    if(COMPUTERE == NULL) {

        printf("A eșuat alocarea memoriei pentru tabloul de structuri.");

    }

    IS\_FREED = 0;

}

int addValues(int i) {

    scanf("%\*c");

    printf("\nCare e modelul computerului %d?\n", i);

    fgets(COMPUTERE[i].modelul, sizeof(COMPUTERE[i].modelul), stdin);

    COMPUTERE[i].modelul[ strlen(COMPUTERE[i].modelul) - 1] = '\0';

    printf("\nCe procesor are computerul %d?\n", i);

    fgets(COMPUTERE[i].procesorul, sizeof(COMPUTERE[i].procesorul), stdin);

    COMPUTERE[i].procesorul[ strlen(COMPUTERE[i].procesorul) - 1] = '\0';

    printf("\nCâtă memorie are computerul %d? (GB)\n", i);

    if(!scanf("%f", &COMPUTERE[i].memoria)) {

        return(0);

    }

    printf("\nCe viteză are computerul %d? (MHz)\n", i);

    if(!scanf("%f", &COMPUTERE[i].viteza)){

        return(0);

    }

    printf("\nCare e prețul computerului %d? (MDL)\n", i);

    if(!scanf("%f", &COMPUTERE[i].pretul)){

        return(0);

    }

    return (1);

}

int introduce(){

    if(!COMPUTERE){

        printf("Tabloului de structuri nu i s-a alocat memorie.\n");

        return(1);

    }

    for (int i = 0; i < SIZE; i++) {

        if(!addValues(i)){

            printf("Valoare invalidă\n");

            return(0);

        }

    }

    IS\_INITIALIZED = 1;

    return(1);

}

int validityCheck(unsigned int i) {

    if(!COMPUTERE && i >= 1){

        printf("Tabloului de structuri nu i s-a alocat memorie.\n");

        return 0;

    }

    if(IS\_FREED && i >= 2) {

        printf("Memoria alocată pentru tablou a fost eliberată, iar operații cu dânsul pot fi imprevizibile.\n");

        return(0);

    }

    if(!IS\_INITIALIZED && i >= 3) {

        printf("Tabloul nu a fost inițializat.\n");

        return 0;

    }

    return 1;

}

void showComputer(int i) {

    printf("\nComputerul %d:\n", i);

    printf("\t1.Modelul %s;\n", COMPUTERE[i].modelul);

    printf("\t2.Procesorul: %s;\n", COMPUTERE[i].procesorul);

    printf("\t3.Memoria: %f GB;\n", COMPUTERE[i].memoria);

    printf("\t4.Viteza: %f MHz;\n", COMPUTERE[i].viteza);

    printf("\t5.Prețul: %f MDL.\n", COMPUTERE[i].pretul);

}

void show() {

    for (int i = 0; i < SIZE; i++) {

        showComputer(i);

    }

}

int push() {

    SIZE++;

    COMPUTERE = (Computer \*)realloc(COMPUTERE, SIZE \* sizeof(Computer));

    if(!addValues(SIZE - 1)){

        printf("Valoare invalidă.\n");

        return(0);

    }

    if(COMPUTERE == NULL) {

        printf("A eșuat realocarea memoriei pentru tabloul de structuri.");

    }

}

int modify() {

    printf("\nElementele tabloului:\n");

    for (int i = 0; i < SIZE; i++) {

        printf("%d: modelul %s\n", i, COMPUTERE[i].modelul);

    }

    int i = 0;

    printf("\nIntroduceți numărul elementului pe care doriți să-l modificați: ");

    if(!scanf("%d", &i)){

        printf("Valoare invalidă.\n");

        return(0);

    }

    if (i >= SIZE || i < 0) {

        printf("Elementul %d nu se află în tablou\n", i);

        return(1);

    }

    showComputer(i);

    int j = 0;

    printf("\nCe proprietate doriți să schimbați? (1-5/sau 0 pentru tot)\n");

    if(!scanf("%d", &j)){

        printf("Valoare invalidă.\n");

        return(0);

    }

    switch(j){

        case 1: scanf("%\*c");

                printf("\nCare e modelul computerului %d?\n", i);

                fgets(COMPUTERE[i].modelul, sizeof(COMPUTERE[i].modelul), stdin);

                COMPUTERE[i].modelul[ strlen(COMPUTERE[i].modelul) - 1] = '\0';

                break;

        case 2: scanf("%\*c");

                printf("\nCe procesor are computerul %d?\n", i);

                fgets(COMPUTERE[i].procesorul, sizeof(COMPUTERE[i].procesorul), stdin);

                COMPUTERE[i].procesorul[ strlen(COMPUTERE[i].procesorul) - 1] = '\0';

                break;

        case 3: scanf("%\*c");

                printf("\nCâtă memorie are computerul %d? (GB)\n", i);

                if(!scanf("%f", &COMPUTERE[i].memoria)) {

                    return(0);

                }

                break;

        case 4: scanf("%\*c");

                printf("\nCe viteză are computerul %d? (MHz)\n", i);

                if(!scanf("%f", &COMPUTERE[i].viteza)){

                    return(0);

                }

                break;

        case 5: scanf("%\*c");

                printf("\nCare e prețul computerului %d? (MDL)\n", i);

                if(!scanf("%f", &COMPUTERE[i].pretul)){

                    return(0);

                }

                break;

        case 0: addValues(i);

                break;

        default: printf("\nNu există proprietatea cu numarul %d\n", j);

    }

    return (1);

}

void modelSearch(char \*cmp) {

    printf("\nComputere de modelul căutat de voi:\n");

    for (int i = 0; i < SIZE; i++) {

        if(strcmp(COMPUTERE[i].modelul, cmp) == 0){

            printf("%d\n", i);

        }

    }

}

void procSearch(char \*cmp) {

    printf("\nComputere cu procesorul căutat de voi:\n");

    for (int i = 0; i < SIZE; i++) {

        if(strcmp(COMPUTERE[i].procesorul, cmp) == 0){

            printf("%d\n", i);

        }

    }

}

void memSearch(float cmp) {

    printf("\nComputere cu memoria căutată de voi:\n");

    for (int i = 0; i < SIZE; i++) {

        if(COMPUTERE[i].memoria == cmp){

            printf("%d\n", i);

        }

    }

}

void spdSearch(float cmp) {

    printf("\nComputere cu viteza căutată de voi:\n");

    for (int i = 0; i < SIZE; i++) {

        if(COMPUTERE[i].viteza == cmp){

            printf("%d\n", i);

        }

    }

}

void priceSearch(float cmp) {

    printf("\nComputere cu prețul căutat de voi:\n");

    for (int i = 0; i < SIZE; i++) {

        if(COMPUTERE[i].pretul == cmp){

            printf("%d\n", i);

        }

    }

}

int search() {

    printf("\nProprietăți disponibile: \n");

    printf("1.modelul\n2.procesorul\n3.memoria\n4.viteza\n5.pretul\n");

    printf("\nPrin care proprietate doriți să căutați?\n");

    int i;

    if(!scanf("%d", &i)){

        printf("Valoare invalidă.\n");

        return(0);

    }

    char scmp[100];

    float fcmp;

    scanf("%\*c");

    switch(i){

        case 1: printf("\nCe model doriți să căutați?\n");

                fgets(scmp, sizeof(scmp), stdin);

                scmp[ strlen(scmp) - 1] = '\0';

                modelSearch(scmp);

            break;

        case 2: printf("\nCe procesor doriți să căutați?\n");

                fgets(scmp, sizeof(scmp), stdin);

                scmp[ strlen(scmp) - 1] = '\0';

                procSearch(scmp);

            break;

        case 3: printf("\nCe memorie doriți să căutați?\n");

                if(!scanf("%f", &fcmp)){

                    printf("Valoare invalidă.\n");

                    return(0);

                }

                memSearch(fcmp);

            break;

        case 4: printf("\nCe viteză doriți să căutați?\n");

                if(!scanf("%f", &fcmp)){

                    printf("Valoare invalidă.\n");

                    return(0);

                }

                spdSearch(fcmp);

            break;

        case 5: printf("\nCe preț doriți să căutați?\n");

                if(!scanf("%f", &fcmp)){

                    printf("Valoare invalidă.\n");

                    return(0);

                }

                priceSearch(fcmp);

            break;

        default: printf("Proprietatea cu numărul dat nu există.\n");

    }

    return(1);

}

void modelSort() {

    Computer tmp;

    for (int i = 0; i < SIZE; i++) {

        for (int j = i + 1; j < SIZE; j++) {

            if(strcmp(COMPUTERE[i].modelul, COMPUTERE[j].modelul) > 0) {

                tmp = COMPUTERE[i];

                COMPUTERE[i] = COMPUTERE[j];

                COMPUTERE[j] = tmp;

            }

        }

    }

}

void procSort() {

    Computer tmp;

    for (int i = 0; i < SIZE; i++) {

        for (int j = i + 1; j < SIZE; j++) {

            if(strcmp(COMPUTERE[i].procesorul, COMPUTERE[j].procesorul) > 0) {

                tmp = COMPUTERE[i];

                COMPUTERE[i] = COMPUTERE[j];

                COMPUTERE[j] = tmp;

            }

        }

    }

}

void memSort() {

    Computer tmp;

    for (int i = 0; i < SIZE; i++) {

        for (int j = i + 1; j < SIZE; j++) {

            if(COMPUTERE[i].memoria < COMPUTERE[j].memoria) {

                tmp = COMPUTERE[i];

                COMPUTERE[i] = COMPUTERE[j];

                COMPUTERE[j] = tmp;

            }

        }

    }

}

void spdSort() {

    Computer tmp;

    for (int i = 0; i < SIZE; i++) {

        for (int j = i + 1; j < SIZE; j++) {

            if(COMPUTERE[i].viteza < COMPUTERE[j].viteza) {

                tmp = COMPUTERE[i];

                COMPUTERE[i] = COMPUTERE[j];

                COMPUTERE[j] = tmp;

            }

        }

    }

}

void prcSort() {

    Computer tmp;

    for (int i = 0; i < SIZE; i++) {

        for (int j = i + 1; j < SIZE; j++) {

            if(COMPUTERE[i].pretul < COMPUTERE[j].pretul) {

                tmp = COMPUTERE[i];

                COMPUTERE[i] = COMPUTERE[j];

                COMPUTERE[j] = tmp;

            }

        }

    }

}

int sort() {

    printf("\nProprietăți disponibile: \n");

    printf("1.modelul\n2.procesorul\n3.memoria\n4.viteza\n5.pretul\n");

    printf("\nDupă ce proprietate doriți să sortați?\n");

    int i;

    if(!scanf("%d", &i)){

        printf("Valoare invalidă.\n");

        return(0);

    }

    scanf("%\*c");

    switch(i){

        case 1: modelSort();

            break;

        case 2: procSort();

            break;

        case 3: memSort();

            break;

        case 4: spdSort();

            break;

        case 5: prcSort();

            break;

        default: printf("Proprietatea cu numărul dat nu există.\n");

    }

    return(1);

}

int delete() {

    show();

    printf("\nCe element ați dori sa eliminați?\n");

    int del;

    if(!scanf("%d", &del) || del >= SIZE || del < 0){

        printf("Valoare invalidă.\n");

        return(0);

    }

    Computer \*tmp;

    tmp = (Computer \*)malloc((SIZE - 1) \* sizeof(Computer));

    if(tmp == NULL) {

        printf("A eșuat alocarea memoriei pentru noul tablou de structuri.");

    }

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

        if(i != del){

            tmp[j] = COMPUTERE[i];

            j++;

        }

    }

    SIZE--;

    COMPUTERE = NULL;

    COMPUTERE = tmp;

}

int handleOptions(int in) {

    switch(in) {

        case 1: computerAlloc();

                if(COMPUTERE == NULL){

                    return(0);

                }

                break;

        case 2: if (validityCheck(2)){

                    if(!introduce()){

                        return(0);

                    }

                }

                break;

        case 3: if(validityCheck(3)) {

                    show();

                }

                break;

        case 4: if(validityCheck(3)){

                    if(!push()) {

                        return (0);

                    }

                }

                break;

        case 5: if(validityCheck(3)){

                    if(!modify()) {

                        return (0);

                    }

                }

                break;

        case 6: if(validityCheck(3)) {

                   if(!search()) {

                       return(0);

                   }

                }

                break;

        case 7: if(validityCheck(3)) {

                    if(!sort()) {

                        return(0);

                    }

                }

                break;

        case 8: if(validityCheck(3)) {

                    if(!delete()) {

                        return(0);

                    }

                }

                break;

        case 9: if(validityCheck(2)) {

                    free(COMPUTERE);

                    IS\_FREED = 1;

                    IS\_INITIALIZED = 0;

                }

                break;

        case 0:

            return 1;

        default:printf("Valoarea %d nu există în meniul de opțiuni, introduceți o altă valoare.\n", in);

    }

    return (1);

}

int evaluate() {

    int in;

    do{

        showOptions();

        printf("\nCe operațiune doriți să efectuați?\n");

        if(!scanf("%d", &in)){

            printf("Valoare invalidă.\n");

            return (0);

        }

        else if(!handleOptions(in)){

            return(0);

        }

    }  while(in != 0);

    return (1);

}

int main () {

    if (!evaluate()){

        return(1);

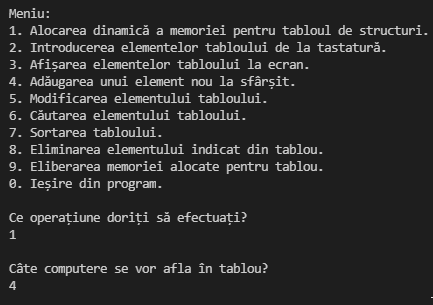
    }

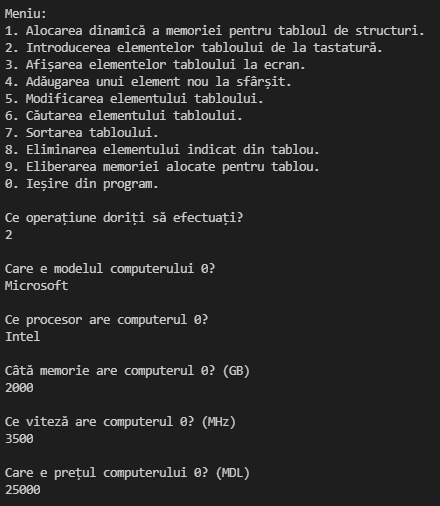
    return (0);

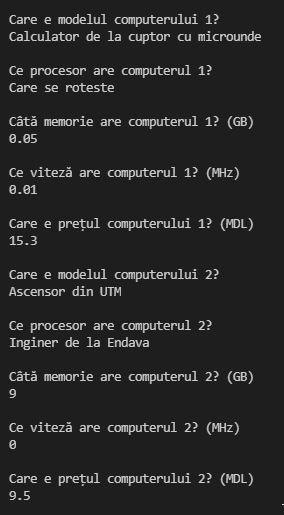
}

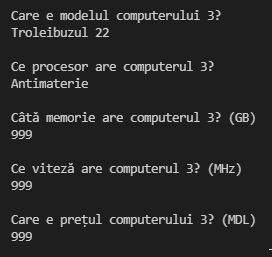
**Execuția programului:**

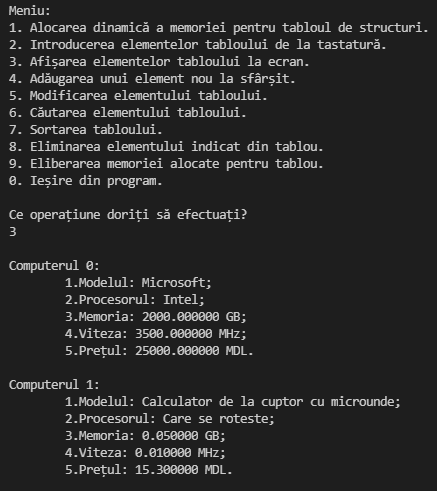
1. Set de date nr.1 :

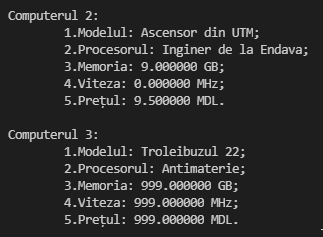


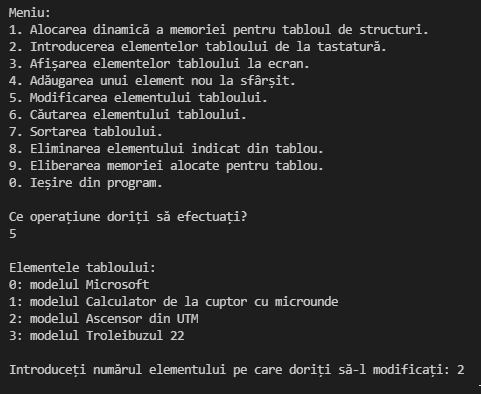


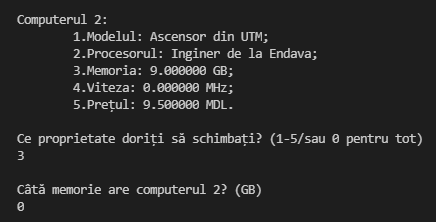


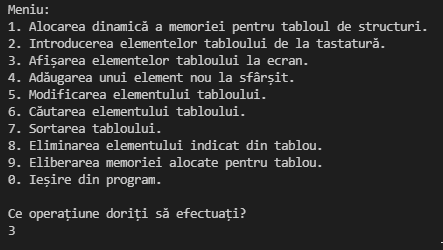


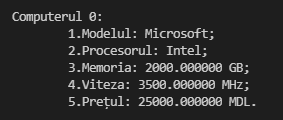


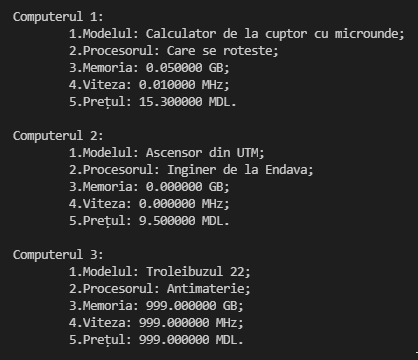


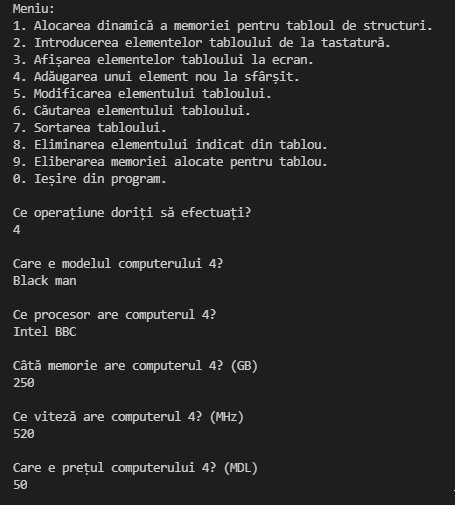


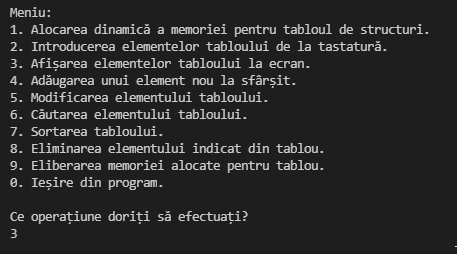


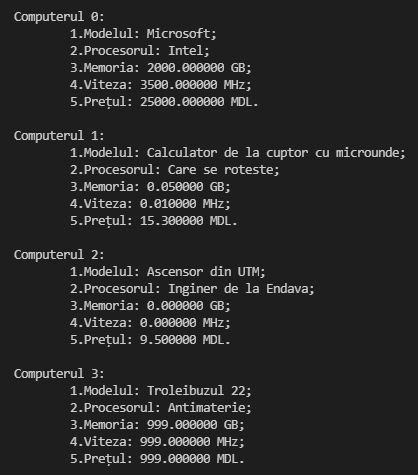


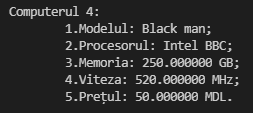


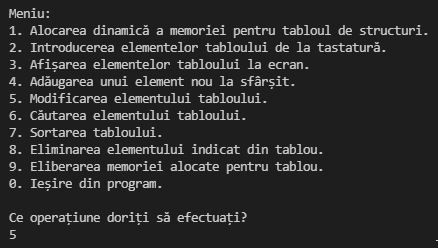


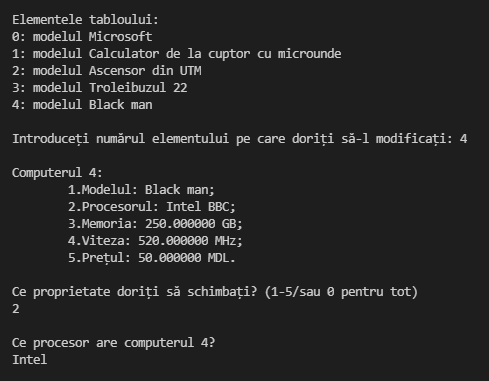


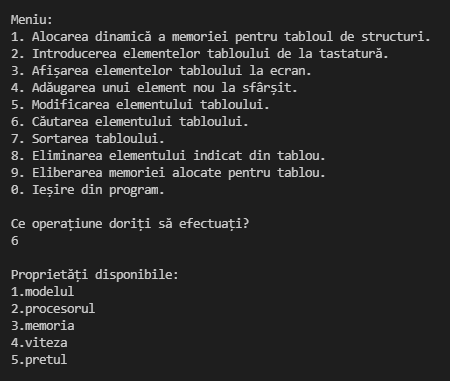


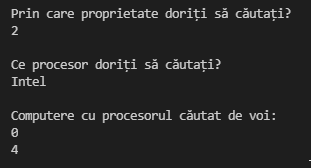


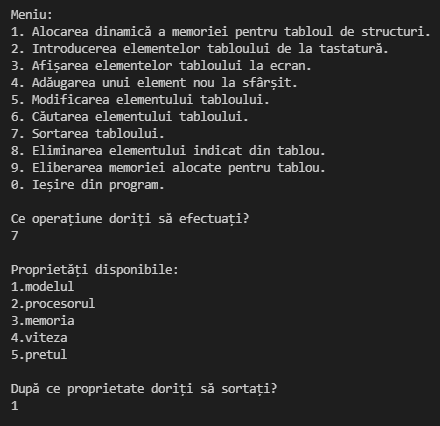


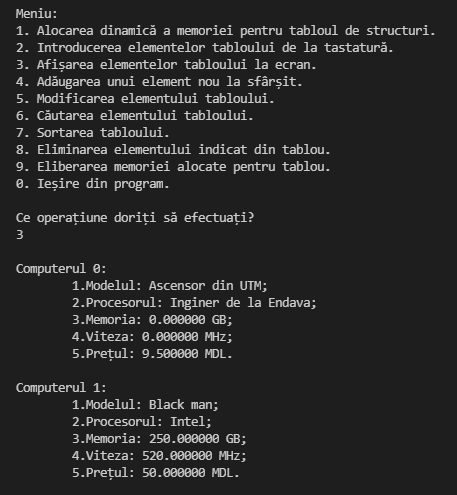


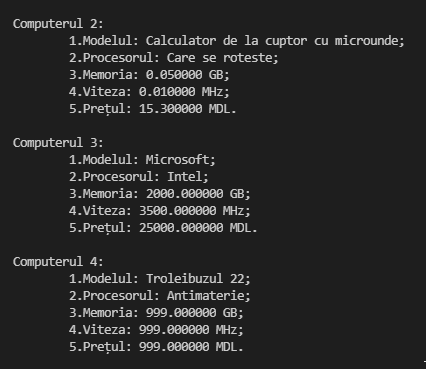


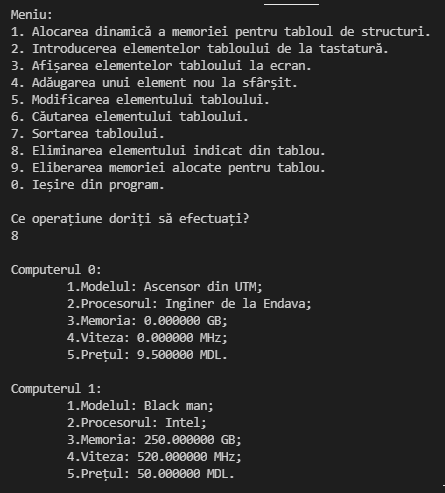


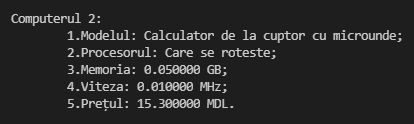


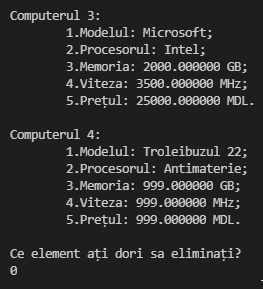


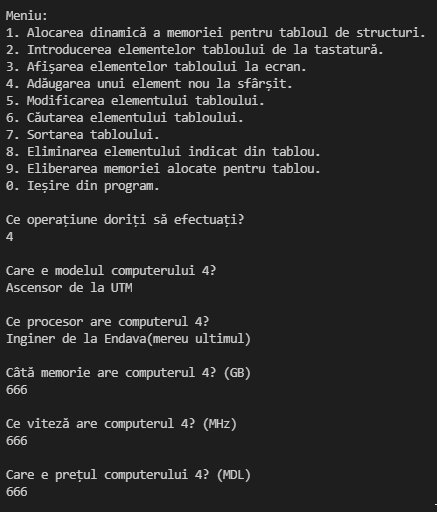


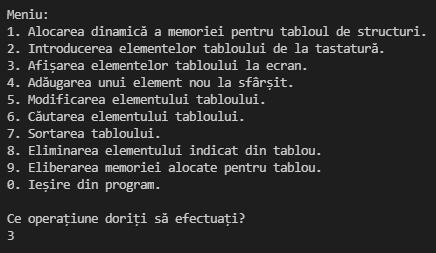


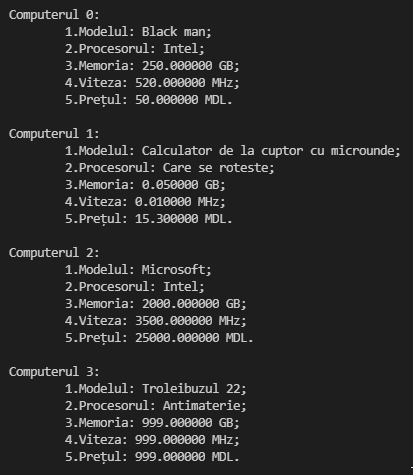


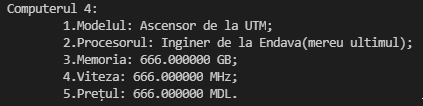


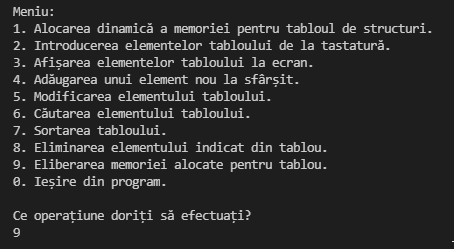


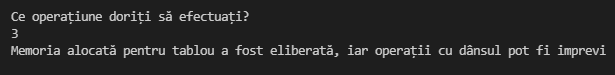






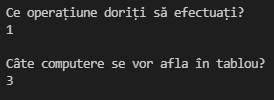


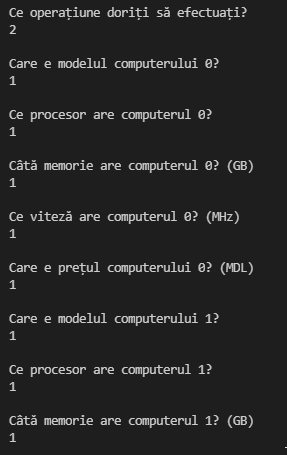


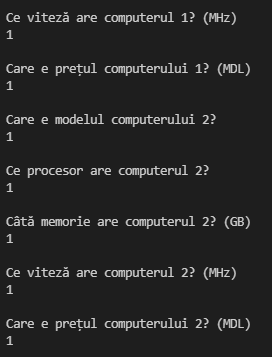


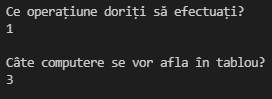
Pentru brevitate, de aici în colo nu se va mai arăta meniul.

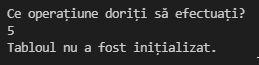
1. Set de date nr.2:



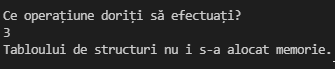


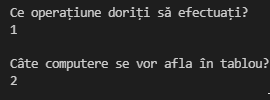


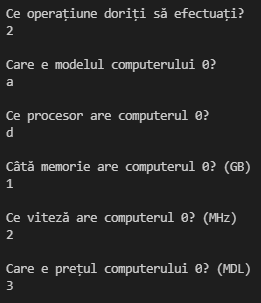


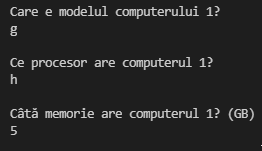


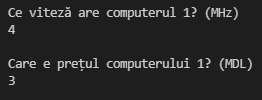
1. Set de date nr.3:

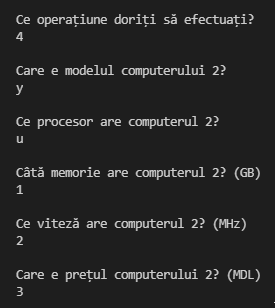


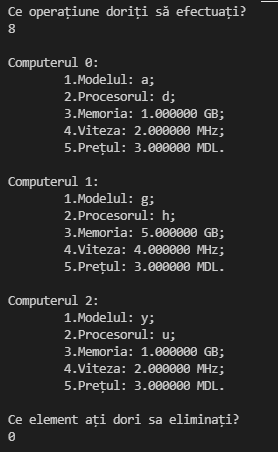


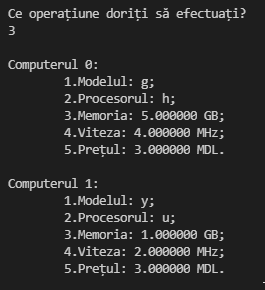




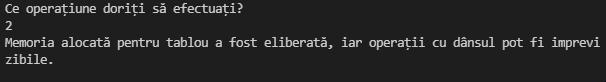












****

După cum se observă, codul reacționează adecvat la orice set de date, evitând orice eroare sau manipulare incorectă a memoriei alocate dinamic. În caz că operațiunile efectuate de utilizator au putea duce la erori, atunci programul nu le efectuază și afișează un mesaj care explică problema și cauza ei.

**Concluzii:**

1. Structurile sunt un mod organizat de a manipula mai multe variabile, posibil și de diferite tipuri, care, totuși, au o legătură comună și ar avea sens să fie manipulate împreună.
2. Structurile nu pot fi copiate direct una de la alta, un mod valid de a opera cu ele fiind prin adresa lor, fie oferind-o unei funcții, sau atribuind-o altei structuri de același tip. În programul meu, eu am decis să fac tabloul meu de structuri o variabilă globală, deoarece toate funcțiile din program se referă în primul rând la el, ci nu la mai multe tablouri.
3. Prin tablouri de structuri se pot organiza mai multe instanțe ale aceleiași structuri pentru a face operațiunile cu dânsele mult mai intuitive. Spre exemplu, dacă există o structură “Legumă”, atunci ar fi mai comod ca instanțe de Legumă, precum “Morcov”, “Cartof”, “Roșie” etc. să se afle într-un tablou comun.

**Surse:**

* **cplusplus.com, Cursul de programare in C++, Structuri de date:**

https://www.cplusplus.com/doc/tutorial-ro/structures/