**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**RAPORT**

Lucrare de laborator nr. 4

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

**Tema: Structuri dinamice de date**

A efectuat :  **St. gr. CR-221FR Serba Cristina**

**A verificat: Asis.univ. Toma Olga**

**Chișinău 2023**

**Cuprins**

[INTRODUCERE 2](#_Toc1325719330)

[REALIZAREA PRACTICĂ A SARCINII DE LUCRU 3](#_Toc126945356)

[CONCLUZII 8](#_Toc1084661404)

[BIBLIOGRAFIE 8](#_Toc1860109079)

# 

# INTRODUCERE

În programarea în limbajul C, o listă legată (linked list) reprezintă o structură de date flexibilă și dinamică, utilizată pentru a stoca și organiza informații. O listă este o colecție de noduri, unde fiecare nod conține date și o referință către următorul nod din listă. Diferența fundamentală între o legătură și alte structuri de date precum tablourile este aceea că o listă legată nu necesită alocarea continuă de memorie. În schimb, fiecare nod din legătură poate fi stocat într-o zonă de memorie arbitrară, iar conexiunile între noduri sunt realizate prin intermediul referințelor. Fiecare nod dintr-o legătură este format din două componente principale: o parte de date, care poate stoca informații diverse (de exemplu, numere întregi, caractere sau structuri mai complexe), și o referință (sau un pointer) către următorul nod din legătură. Ultimul nod dintr-o legătură are o referință nulă, semnificând sfârșitul listei.

Luând în considerare acest fapt, în programul următor vor fi efectuate parcurgeri totale ale listei date pentru a fi prelucrate elementele acesteea.

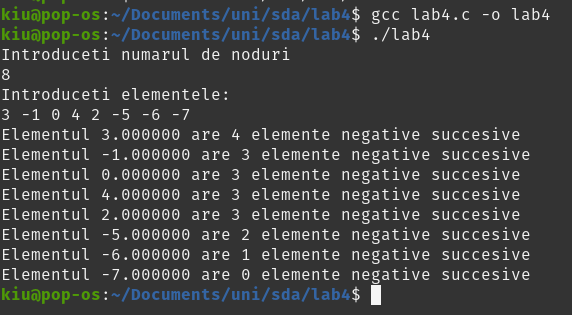
# REALIZAREA PRACTICĂ A SARCINII DE LUCRU

**Sarcina lucrării de laborator (varianta 13):**

Fie dată o listă de numere reale. Pentru fiecare element al listei să se tipărească numărul elementelor negative, ce-l succed.

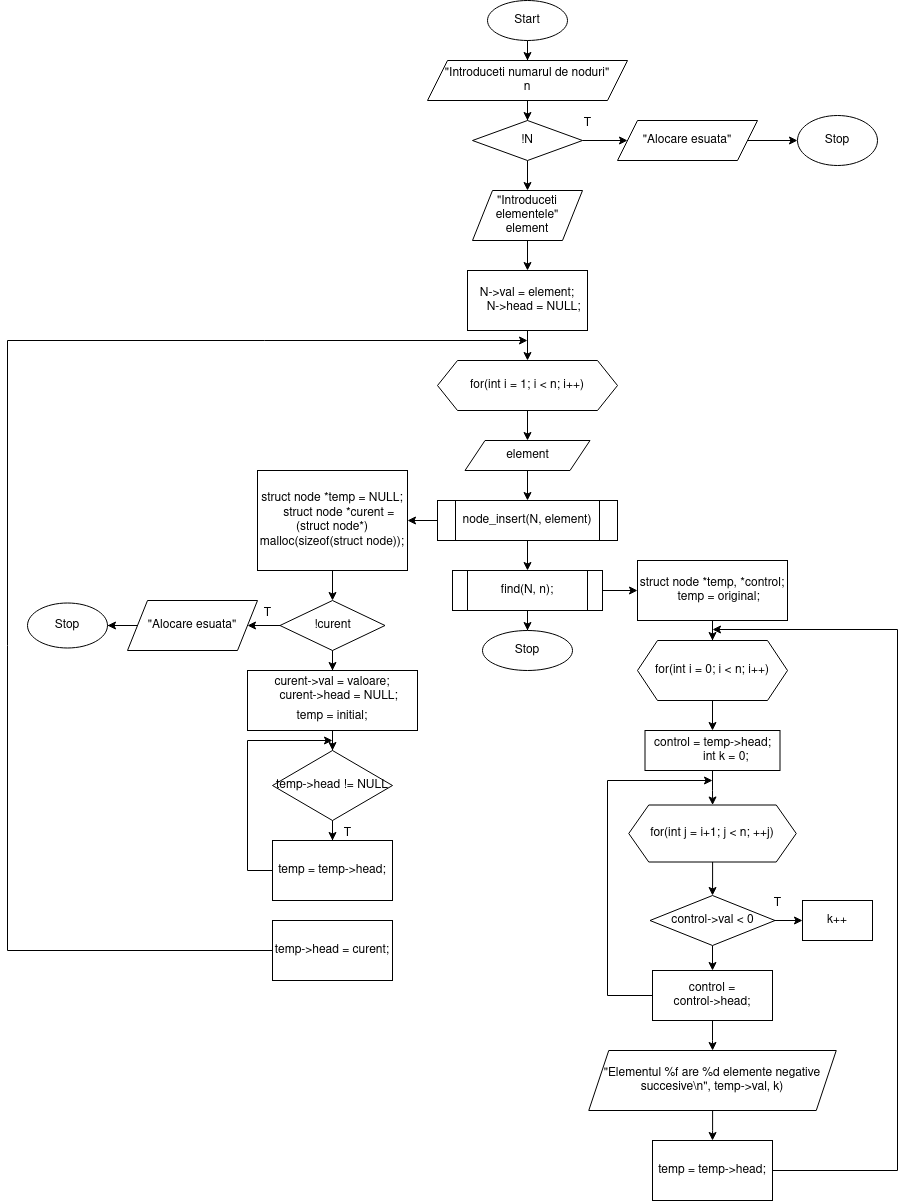
|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  /\* Fie dată o listă de numere reale. Pentru fiecare element al listei  să se tipărească numărul elementelor negative, ce-l succed. \*/  struct node  {  float val;  struct node \*head;  };  void node\_insert(struct node \*initial, int valoare)  {  struct node \*temp = NULL;  struct node \*curent = (struct node\*) malloc(sizeof(struct node));  if(!curent)  {  printf("Alocare esuata\n");  exit(1);  }  curent->val = valoare;  curent->head = NULL;  //temp si initial vor avea aceeasi adresa in memorie  temp = initial;  while(temp->head != NULL)  {  temp = temp->head; //trece prin toate nodurile pana ajunge la ultimul  }  temp->head = curent; //adauga noul nod la capat  }  void find(struct node \*original, int n)  {  struct node \*temp, \*control;  temp = original;  for(int i = 0; i < n; i++)  {  control = temp->head;  int k = 0;  //pentru fiecare nod va trece prin intreaga lista  //incepand de la urmatorul  for(int j = i+1; j < n; ++j)  {  if(control->val < 0) k++;  control = control->head;  }  printf("Elementul %f are %d elemente negative succesive\n", temp->val, k);  temp = temp->head;  }  }  int main()  {  printf("Introduceti numarul de noduri\n");  int n;  scanf("%d", &n);  struct node \*N = (struct node\*) malloc(sizeof(struct node));  if(!N)  {  printf("Alocare esuata\n");  exit(1);  }  float element;  printf("Introduceti elementele: \n");  //-------------------------  //pentru primul nod  scanf("%f", &element);  N->val = element;  N->head = NULL;  //-------------------------  for(int i = 1; i < n; i++)  {  scanf("%f", &element);  node\_insert(N, element);  }  find(N, n);  return 0;  } |

Programul inițializează lista înlățuită prin funția inserare\_nod. Aceasta primește ca parametri un nod inițial și o valoare. Se creează un nou nod și se atribuie valoarea dată. Apoi, se parcurge lista până la ultimul nod și se adaugă noul nod la capătul listei. La adăugarea tuturor elementelor dorite în listă, funcția find primește ca parametri un nod inițial și numărul total de noduri din listă. Aceasta parcurge fiecare nod din listă și pentru fiecare nod, calculează numărul de elemente negative succesive prin parcurgerea listei începând de la următorul nod. Acest lucru este efectuat prin condiția if unde fiecare nod succesiv este evaluat, iar în caz de adevăr, variabila control k va fi incrementată, păstrând, în același timp valoarea nodului curent.



**Fig.1.1 *Rezultatul execuției***

Figura 1.1 prezintă rezultatul executiei programului



**Fig.1.2 *Schema bloc a programului***

Figura 1.2 prezintă schema bloc a programului

# CONCLUZII

Programul prezentat demonstrează implementarea și utilizarea listelor înlănțuite în limbajul C pentru a număra și afișa numărul de elemente negative succesive pentru fiecare element al unei liste.

Utilizarea listelor înlănțuite în acest context oferă avantaje semnificative, cum ar fi flexibilitatea în gestionarea și manipularea datelor, posibilitatea de a adăuga și elimina noduri în mod dinamic și economia de memorie prin alocarea doar a spațiului necesar.

Programul ilustrează, de asemenea, utilizarea pointerilor și alocarea dinamică a memoriei pentru a crea și manipula lista. Prin intermediul acestui program, am învățat cum să inserez noduri într-o listă și cum să parcurg lista pentru a realiza operațiile necesare. Am înțeles, de asemenea, modul în care putem utiliza pointerii pentru a accesa și a manipula nodurile din listă.

În concluzie, prin intermediul acestui program, am demonstrat aplicarea practică a conceptelor legate de listele înlănțuite în limbajul C.

# BIBLIOGRAFIE

[***https://app.diagrams.net/***](https://app.diagrams.net/)