Ministerul Educaţiei și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi Microelectronică

**RAPORT**

Lucrare de laborator Nr.4

*la Structuri de Date și Algoritmi*

Tema: Algoritmi de prelucrare a listelor liniare simplu

înlănțuite (listelor unidirecționale)

A efectuat: st. gr. SI-212 Boboc Sandu

A verificat: lect. asist. Mititelu Vitalii

Chişinău 2022

**Scopul:**

Obținerea deprinderilor practice de implementare și de

utilizare a tipului abstract de date (TAD) „Listă simplu înlănțuită” în

limbajul C cu asigurarea operațiilor de prelucrare de bază ale listei.

**Sarcina:**

Să se scrie 3 fișiere-text în limbajul C pentru implementarea și utilizarea TAD

„Listă simplu înlănțuită” cu asigurarea operațiilor de prelucrare de bază ale listei:

1. Fișier antet cu extensia .h, care conține specificarea structurii de date a

elementului listei simplu înlănțuite (conform variantelor) și prototipurile

funcțiilor de prelucrare de bază ale listei.

2. Fișier cu extensia .c sau .cpp, care conține implementările (codurile)

funcțiilor declarate în fișierul antet.

3. Fișier al utilizatorului, funcția mаin() pentru prelucrarea listei cu

afișarea la ecran a următorului meniu de opțiuni de bază:

1. Crearea listei în memoria dinamică

2. Introducerea informației despre elementele listei de la tastatură.

3. Afișarea informației despre elementele listei la ecran.

4. Căutarea elementului în listă.

5. Modificarea câmpurilor unui element din listă.

6. Determinarea adresei ultimului element din listă.

7. Determinarea lungimii listei (numărul de elemente).

8. Interschimbarea a două elemente indicate în listă.

9. Sortarea listei.

10. Eliberarea memoriei alocate pentru listă.

0. Ieșire din program.

**Varianta 2:**

Structura Film cu câmpurile: denumirea, țara, regizorul, genul filmului, anul.

**Rezumat:**

**Liste simplu inlantuite**

Listele simplu inlantuite sunt structuri de date dinamice omogene. Spre deosebire de masive, listele nu sunt alocate ca blocuri omogene de memorie, ci ca elemente separate de memorie. Fiecare nod al listei contine, in afara ce informatia utila, adresa urmatorului element. Aceasta organizare permite numai acces secvential la elementele listei.

Pentru accesarea listei trebuie cunoscuta adresa primului element (numita capul

listei); elementele urmatoare sunt accesate parcurgand lista.

Lista simplu inlantuita poate fi reprezentata grafic astfel:



**1. Structura listei**

Pentru a asigura un grad mai mare de generalitate listei a fost creat un alias pentru datele utile (in cazul nostru un intreg):

// Datele asociate unui

// element dintr-o lista

typedef int Date;

In cazul in care se doreste memorarea unui alt tip de date, trebuie schimbata doar declaratia aliasului Date.

Pentru memorarea listei se foloseste o structura autoreferita. Acesta structura va avea forma:

// Structura unui element

// dintr-o lista simplu inlantuita

struct Element

{

// datele efective memorate

Date valoare;

// legatura catre nodul urmator

Element\* urmator;

};

In cazul in care elemenul este ultimul din lista, pointerul urmator va avea valoarea NULL.

Declararea listei se face sub forma:

// declarare lista vida

Element\* cap = NULL;

**Codul deplin al programului:**

**Fișierul program\_principal.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "includes.h"

Node \*inceput = NULL;

void introduce\_sfarsit()

{

Node \*elem\_nou = malloc(sizeof(Node));

if (elem\_nou == NULL)

{

exit(1);

}

elem\_nou->next = NULL;

if (inceput == NULL)

{

inceput = elem\_nou;

return;

}

Node \*elem\_curent = inceput;

while (elem\_curent->next != NULL)

{

elem\_curent = elem\_curent->next;

}

elem\_curent->next = elem\_nou;

}

void creare\_lista()

{

int i;

printf("\nmarimea listei: ");

if (!scanf("%d", &i))

{

printf("\nvaloare invalida.\n");

return;

}

for (; i > 0; i--)

introduce\_sfarsit();

}

Film introduce\_struct()

{

Film in;

printf("denumirea: ");

scanf("%s", in.denumirea);

printf("tara: ");

scanf("%s", in.tara);

printf("regizorul: ");

scanf("%s", in.regizorul);

printf("genul: ");

scanf("%s", in.genul);

printf("anul: ");

scanf("%d", &in.anul);

return in;

}

void introduce()

{

int i = 0;

for (Node \*tmp = inceput; tmp != NULL; tmp = tmp->next, i++)

{

printf("\nstructura %d\n", i);

tmp->data = introduce\_struct();

}

}

void afisare\_struct(Film in)

{

printf("\ndenumirea: ");

printf("%s", in.denumirea);

printf("\ntara: ");

printf("%s", in.tara);

printf("\nregizorul: ");

printf("%s", in.regizorul);

printf("\ngenul: ");

printf("%s", in.genul);

printf("\nanul: ");

printf("%d", in.anul);

printf("\n");

}

void afisare()

{

int i = 0;

for (Node \*tmp = inceput; tmp != NULL; tmp = tmp->next, i++)

{

printf("\nstructura %d", i);

afisare\_struct(tmp->data);

}

}

void cautare()

{

int alegere;

printf("\ncampuri disponibile:\n1.denumirea\n2.anul\n");

printf("\ncampul cautat: ");

if (!scanf("%d", &alegere) || alegere > 2 || alegere < 1)

{

printf("\nvaloare invalida.\n");

return;

}

afisare();

char cNume[20];

int cAn;

printf("\nvaloare cautata: ");

switch (alegere)

{

case 1:

if (!scanf("%s", cNume))

{

printf("\nvaloare invalida.\n");

return;

}

break;

case 2:

if (!scanf("%d", &cAn))

{

printf("\nvaloare invalida.\n");

return;

}

break;

default:

break;

}

int i = 0;

for (Node \*tmp = inceput; tmp != NULL; tmp = tmp->next, i++)

{

if ((alegere == 1 && strcmp(tmp->data.denumirea, cNume) == 0) ||

(alegere == 2 && tmp->data.anul == cAn))

{

printf("\nfilmul %d", i);

afisare\_struct(tmp->data);

}

}

}

int numara()

{

int n = 0;

for (Node \*elem\_curent = inceput; elem\_curent != NULL; elem\_curent = elem\_curent->next)

{

n++;

}

return n;

}

void modifica()

{

afisare();

printf("\nnumarul filmului care va fi modificat: ");

int alegere;

if (!scanf("%d", &alegere) || alegere >= numara() || alegere < 0)

{

printf("\nvaloare invalida.");

return;

}

int i = 0;

for (Node \*tmp = inceput; tmp != NULL; tmp = tmp->next, i++)

{

if (i == alegere)

tmp->data = introduce\_struct();

}

}

Node \*ultima\_adresa()

{

Node \*tmp = inceput;

for (; tmp->next != NULL; tmp = tmp->next)

;

return tmp;

}

Node \*nod\_la(int index)

{

int i = 0;

for (Node \*tmp = inceput; tmp != NULL; tmp = tmp->next, i++)

{

if (i == index)

return tmp;

}

}

void interschimbare\_struct(Node \*primul, Node \*al\_doilea)

{

Film tmp = primul->data;

primul->data = al\_doilea->data;

al\_doilea->data = tmp;

}

void interschimbare()

{

int a, b;

afisare();

printf("\nstructurile care vor interschimbate: ");

if (!scanf("%d %d", &a, &b) || a >= numara() || a < 0 || b >= numara() || b < 0)

{

printf("\nvaloare invalida.\n");

return;

}

Node \*primul = nod\_la(a);

Node \*al\_doilea = nod\_la(b);

interschimbare\_struct(primul, al\_doilea);

}

void sortare()

{

int alegere;

printf("\ncampuri disponibile:\n1.denumirea\n2.anul\n");

printf("\ncampul sortat: ");

if (!scanf("%d", &alegere) || alegere > 2 || alegere < 1)

{

printf("\nvaloare invalida.\n");

return;

}

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

{

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

{

if ((strcmp(nod\_la(i)->data.denumirea, nod\_la(j)->data.denumirea) > 0 && alegere == 1) ||

(nod\_la(i)->data.anul < nod\_la(j)->data.anul && alegere == 2))

{

interschimbare\_struct(nod\_la(i), nod\_la(j));

}

}

}

}

void eliberare()

{

Node \*elem\_curent = inceput;

while (elem\_curent != NULL)

{

Node \*tmp = elem\_curent;

elem\_curent = elem\_curent->next;

free(tmp);

}

inceput = NULL;

}

**Fișierul includes.h:**

#pragma once

typedef struct Film

{

char denumirea[20];

char tara[20];

char regizorul[20];

char genul[20];

int anul;

} Film;

typedef struct Node

{

Film data;

struct Node \*next;

} Node;

void creare\_lista();

void introduce();

void afisare();

void cautare();

void modifica();

Node \*ultima\_adresa();

int numara();

void interschimbare();

void sortare();

void eliberare();

**Fișierul user.c:**

#include <stdio.h>

#include "includes.h"

void afisare\_options()

{

printf("\nMeniu\n");

printf("1. Crearea listei in memoria dinamica.\n");

printf("2. Introducerea informatiei despre elementele listei de la tastatura.\n");

printf("3. Afisarea informatiei despre elementele listei la ecran.\n");

printf("4. Cautarea elementului in lista.\n");

printf("5. Modificarea campurilor unui element din lista.\n");

printf("6. Determinarea adresei ultimului element din lista.\n");

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

printf("8. Interschimbarea a doua elemente indicate in lista.\n");

printf("9. Sortarearea listei.\n");

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

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

}

int handle\_options(int in)

{

int alegere;

switch (in)

{

case 1:

creare\_lista();

break;

case 2:

introduce();

break;

case 3:

afisare();

break;

case 4:

cautare();

break;

case 5:

modifica();

break;

case 6:

printf("\nadresa ultimului element: %p\n", ultima\_adresa());

break;

case 7:

printf("\n%d elemente se afla in lista.\n", numara());

break;

case 8:

interschimbare();

break;

case 9:

sortare();

break;

case 10:

eliberare();

break;

case 0:

eliberare();

return 1;

default:

printf("Valoarea %d nu exista in meniul de optiuni, introduceti o alta valoare.\n", in);

}

return (1);

}

int main()

{

int in;

do

{

afisare\_options();

printf("\nCe operatiune doriti sa efectuati?\n");

if (!scanf("%d", &in))

{

printf("\nvaloare invalida.\n");

return (1);

}

else if (!handle\_options(in))

{

return (1);

}

} while (in != 0);

return (0);

}

**Execuția programului:**





























**Concluzii:**

* Listele simplu înlănțuite prezintă un avantaj față de array-ur prin faptul ca dimensiunea și ordinea elementelor lor poate fi schimbată cu ușurință.
* Pe când într-un array există opțiunea de a seta o dimensiune fixă, care ar putea ocupa mult spațiu fără sens, sau de a folosi malloc, care în cazul maririi unui array mare cu un element consumă foarte multe resurse doar pentru a copia elementele într-un bloc de memorie cu dimensiunea corespunzătoare, listele înlănțuite nu suferă de așa probleme.
* Avantajele listei înlănțuite sunt un rezultat al faptului că, spre deosebire de array-uri, ele nu se află într-un bloc continuu de memorie, ci fiecare element deține adresa către următorul, oferind flexibilitate sporită.

**Surse:**

* softmentor.ro, Cursul de programare in C, Tema: Liste simplu inlantuite: <http://ase.softmentor.ro/StructuriDeDate/Fisiere/03_ListeSimple.pdf>