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 Șeremet Alexandru

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 15:**

Structura Computer cu câmpurile: modelul, procesorul, memoria,

viteza, prețul.

**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 code.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "includes.h"

Node \*root = NULL;

void insert\_end()

{

Node \*new\_node = malloc(sizeof(Node));

if (new\_node == NULL)

{

exit(1);

}

new\_node->next = NULL;

if (root == NULL)

{

root = new\_node;

return;

}

Node \*curr = root;

while (curr->next != NULL)

{

curr = curr->next;

}

curr->next = new\_node;

}

void create\_list()

{

int i;

printf("\nmarimea listei: ");

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

{

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

return;

}

for (; i > 0; i--)

insert\_end();

}

Computer introduce\_struct()

{

Computer in;

printf("modelul: ");

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

printf("procesorul: ");

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

printf("memoria: ");

scanf("%f", &in.memoria);

printf("viteza: ");

scanf("%f", &in.viteza);

printf("pretul: ");

scanf("%f", &in.pretul);

return in;

}

void introduce()

{

int i = 0;

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

{

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

tmp->data = introduce\_struct();

}

}

void show\_struct(Computer in)

{

printf("\nmodelul: ");

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

printf("\nprocesorul: ");

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

printf("\nmemoria: ");

printf("%f", in.memoria);

printf("\nviteza: ");

printf("%f", in.viteza);

printf("\npretul: ");

printf("%f", in.pretul);

printf("\n");

}

void show()

{

int i = 0;

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

{

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

show\_struct(tmp->data);

}

}

void search()

{

int choice;

printf("\ncampuri disponibile:\n1.modelul\n2.pretul\n");

printf("\ncampul cautat: ");

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

{

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

return;

}

show();

char sModel[MAX\_STRLEN];

float sPrice;

printf("\nvaloare cautata: ");

switch (choice)

{

case 1:

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

{

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

return;

}

break;

case 2:

if (!scanf("%f", &sPrice))

{

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

return;

}

break;

default:

break;

}

int i = 0;

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

{

if ((choice == 1 && strcmp(tmp->data.modelul, sModel) == 0) ||

(choice == 2 && tmp->data.pretul == sPrice))

{

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

show\_struct(tmp->data);

}

}

}

int count()

{

int c = 0;

for (Node \*curr = root; curr != NULL; curr = curr->next)

{

c++;

}

return c;

}

void modify()

{

show();

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

int choice;

if (!scanf("%d", &choice) || choice >= count() || choice < 0)

{

printf("\nvaloare invalida.");

return;

}

int i = 0;

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

{

if (i == choice)

tmp->data = introduce\_struct();

}

}

Node \*get\_last\_address()

{

Node \*tmp = root;

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

;

return tmp;

}

Node \*get\_node(int index)

{

int i = 0;

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

{

if (i == index)

return tmp;

}

}

void swap\_struct(Node \*first, Node \*second)

{

Computer tmp = first->data;

first->data = second->data;

second->data = tmp;

}

void swap()

{

int a, b;

show();

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

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

{

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

return;

}

Node \*first = get\_node(a);

Node \*second = get\_node(b);

swap\_struct(first, second);

}

void sort()

{

int choice;

printf("\ncampuri disponibile:\n1.modelul\n2.pretul\n");

printf("\ncampul sortat: ");

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

{

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

return;

}

Computer replace;

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

{

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

{

if ((strcmp(get\_node(i)->data.modelul, get\_node(j)->data.modelul) > 0 && choice == 1) ||

(get\_node(i)->data.pretul < get\_node(j)->data.pretul && choice == 2))

{

swap\_struct(get\_node(i), get\_node(j));

}

}

}

}

void free\_list()

{

Node \*curr = root;

while (curr != NULL)

{

Node \*aux = curr;

curr = curr->next;

free(aux);

}

root = NULL;

}

**Fișierul includes.h:**

#pragma once

#define MAX\_STRLEN 20

typedef struct

{

char modelul[MAX\_STRLEN];

char procesorul[MAX\_STRLEN];

float memoria;

float viteza;

float pretul;

} Computer;

typedef struct Node

{

Computer data;

struct Node \*next;

} Node;

void create\_list();

void introduce();

void show();

void search();

void modify();

Node \*get\_last\_address();

int count();

void swap();

void sort();

void free\_list();

**Fișierul user.c:**

#include <stdio.h>

#include "includes.h"

void show\_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. Sortarea listei.\n");

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

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

}

int handle\_options(int in)

{

int choice;

switch (in)

{

case 1:

create\_list();

break;

case 2:

introduce();

break;

case 3:

show();

break;

case 4:

search();

break;

case 5:

modify();

break;

case 6:

printf("\nadresa ultimului element: %p\n", get\_last\_address());

break;

case 7:

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

break;

case 8:

swap();

break;

case 9:

sort();

break;

case 10:

free\_list();

break;

case 0:

free\_list();

return 1;

default:

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

}

return (1);

}

int handle\_input()

{

int in;

do

{

show\_options();

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

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

{

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

return (0);

}

else if (!handle\_options(in))

{

return (0);

}

} while (in != 0);

return (1);

}

int main()

{

if (!handle\_input())

{

return (1);

}

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>