Universitatea Tehnică a Moldovei

Facultatea *Calculatoare, Informatică și Microelectronică*

Specialitatea *Tehnologii Informaționale*

Raport

la lucrarea de laborator nr. 6

Tema: *"Algoritmi de prelucrare a listelor liniare simplu înlănțuite (listelor unidirecționale)"*

Disciplina: "Structuri de date și algoritmi"

Varianta 4

A efectuat:	Student grupa TI-231 FR	Apareci Aurica
A verificat:	Asistent universitar	Mantaluță Marius

Chișinău 2024

Cuprins

1. Cadrul teoretic

Scopul: Obținerea deprinderilor practice de implementare și de utilizare a tipului abstract de date „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 tipului abstract de date „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, care conține implementările funcțiilor declarate în fișierul antet.

3. Fișier al utilizatorului, funcția main() pentru prelucrarea listei cu afișarea la ecran a următorului meniu de opțiuni de bază:

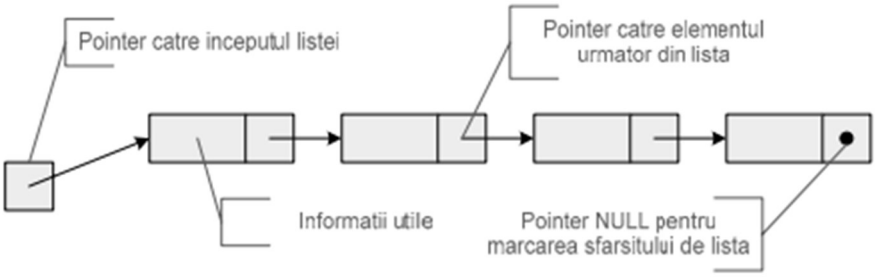
1. Crearea listei în memoria dinamică
2. Introducerea informației despre elementele listei de la tastatură.
3. Afișarea informației despre elementele listei la ecran.
4. Căutarea elementului în listă.
5. Modificarea câmpurilor unui element din listă.
6. Determinarea adresei ultimului element din listă.
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 4 Structura *Farmacie* cu câmpurile: denumirea, adresa, telefonul, orele de lucru, volumul de medicamente.

2. Repere teoretice

Listele simplu înlănțuite 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 conține, în afara ce informația utilă, adresa urmatorului element. Această organizare permite numai acces secvențial la elementele listei. Pentru accesarea listei trebuie cunoscută adresa primului element (numita capul listei); elementele urmatoare sunt accesate parcurgand listă.

Lista simplu înlănțuită poate fi reprezentată grafic astfel:



3. Listingul programului

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "user.h"

int main(){
    while (go)
    {
        userChose = Menu();
        BL();
    }
    return 0;
}
```

struct.h

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Farmacie{
    char denumire[100];
    char adresa[100];
    char telefon[100];
    char oreLucru[100];
    int volumMedicamente;
} Farmacie;

Farmacie * ReadStruct(){
    Farmacie * c = (Farmacie * )malloc(sizeof(Farmacie));
    printf("Dati datele Farmaciei:\n==============================\n");
    printf("Denumire:\t");
    fflush(stdin);
    gets(c->denumire);
    printf("Adresa:\t");
    fflush(stdin);
    gets(c->adresa);
    printf("Telefon:\t");
    fflush(stdin);
    gets(c->telefon);
    printf("Ore de lucru:\t");
    fflush(stdin);
    gets(c->oreLucru);
    printf("Volum medicamente:\t");
    fflush(stdin);
    scanf("%d", &c->volumMedicamente);
    return c;
}
```

```
void ShowStruct(Farmacie * c, int id){
    if (c==NULL)
    {
        printf("null reference exception");
        return;
    }
    printf("==================================\n");
    printf("Farmacie %d:\n", id);
    printf("Denumire:\t%s\n", c->denumire);
    printf("Adresa:\t%s\n", c->adresa);
    printf("Telefon:\t%s\n", c->telefon);
    printf("Ore de lucru:\t%s\n", c->oreLucru);
    printf("Volum medicamente:\t%d\n", c->volumMedicamente);
    printf("==================================\n");
}

char * ToString(Farmacie * c){
    char * str = (char *)malloc(1000);
    sprintf(str, "%s_|_%s_|_%s_|_%s_|_%d\n", c->denumire, c->adresa, c->telefon, c-
>oreLucru, c->volumMedicamente);
    return str;
}
Farmacie * FromString(char * str){
    Farmacie * c = (Farmacie *)malloc(sizeof(Farmacie));
    sscanf(str, "%[^_]_|_%[^_]_|_%[^_]_|_%[^_]_|_%d\n", c->denumire, c->adresa, c-
>telefon, c->oreLucru, &c->volumMedicamente);
    return c;
}

void ModifyStruct(Farmacie * c){
    char resp;
    printf("Modificati denumirea farmaciei?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp)=='y')
    {
        fflush(stdin);
        printf("Denumirea farmaciei:\t");
        gets(c->denumire);
    }
    printf("Modificati adresa farmaciei?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp)=='y')
    {
        fflush(stdin);
        printf("Adresa farmaciei:\t");
        gets(c->adresa);
    }
    printf("Modificati telefonul farmaciei?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp)=='y')
    {
        fflush(stdin);
```

```
        printf("Telefonul farmaciei:\t");
        gets(c->telefon);
    }
    printf("Modificati orele de lucru?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp)=='y')
    {
        fflush(stdin);
        printf("Orele de lucru:\t");
        gets(c->oreLucru);
    }
    printf("Modificati volumul de medicamente?(y/n) ->");
    fflush(stdin);
    scanf("%c", &resp);
    if (tolower(resp)=='y')
    {
        fflush(stdin);
        printf("Volumul de medicamente:\t");
        scanf("%d", &c->volumMedicamente);
    }
}
```

user.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <conio.h>
#include "struct.h"
#include "list.h"

List * lista;
void Read(){
    if (lista==NULL)
    {
        lista = new_list();
    }
    char rs = 'y';
    do
    {
        Farmacie * f = ReadStruct();
        push_first(lista,f)?
            printf("succes\n"):
            printf("error\n");
        printf("Mai introduceti Farmacii(y/n)? : ");
        fflush(stdin);
        scanf("%c", &rs);
        rs = tolower(rs);
    } while (rs == 'y');
}

void ShowList(){
    if (lista==NULL)
    {
        printf("Lista nula\n");
```

```
        return;
    }
    if (is_empty(lista))
    {
        printf("Lista goala");
        return;
    }
    Node * curent_node = lista->head;
    while (curent_node!=NULL)
    {
        ShowStruct(curent_node->value,curent_node->index);
        curent_node = curent_node->next;
    }
}

void Search(){
    if (lista==NULL)
    {
        printf("Lista nula\n");
        return;
    }
    printf("Introduceti Numele farmaciei:");
    fflush(stdin);
    char mod[100];
    gets(mod);
    Node * curent_node = lista->head;
    while (curent_node!=NULL)
    {
        if (strcmp(((struct Farmacie *)(curent_node->value))->denumire, mod)==0)
        {
            ShowStruct(curent_node->value,curent_node->index);
            return;
        }
        curent_node = curent_node->next;
    }
    printf("Elementul cautat nu se afla in lista\n");
}

void SortList(){
    struct Node * current = lista->head,
                * index = NULL;
    Farmacie * temp;
    if (lista==NULL)
    {
        printf("Lista nula\n");
        return;
    }
    if (is_empty(lista))
    {
        printf("Lista goala");
        return;
    }
    while(current != NULL)
    {
        index = current->next;
```

```
        while(index != NULL)
        {
            if(((struct Farmacie *)(current->value))->denumire[0] >
                    ((struct Farmacie *)(index->value))->denumire[0])
            {
                temp = current->value;
                current->value = index->value;
                index->value = temp;
            }
            index = index->next;
        }
        current = current->next;
    }
    printf("Lista a fost sortata cu succes in ordine alfabetica dupa numele
farmaciilor\n");
}

void Update(){
    printf("Introduceti ID-ul Farnaciei ce urmeaza a fi actualizata: ");
    int ID = 0;
    scanf("%d", &ID);
    if (ID>lista->Count-1)
    {
        printf("Ivalid index\n");
        return;
    }
        Node * curent = lista->head;
        while (curent!=NULL)
        {
            if (ID==curent->index)
            {
                break;
            }
            curent = curent->next;
        }
        ModifyStruct((Farmacie*)curent->value);
        curent = NULL;
        printf("Datele datele au fost actualziate\n");
}

void LastIndex(){
    Node * curent = lista->head;
    while (curent->next!=NULL)
    {
        curent = curent->next;
    }
    printf("Ultimul nod se afla pe pozitia %d si are adresa in memorie: %p", curent-
>index,&curent);
}

void swapNodes(int x, int y){
    if (x == y)
        return;

    Node * prevX = NULL, *currX = lista->head;
```

```
    while (currX != NULL && currX->index != x) {
        prevX = currX;
        currX = currX->next;
    }

    Node * prevY = NULL,* currY = lista->head;
    while (currY != NULL && currY->index != y) {
        prevY = currY;
        currY = currY->next;
    }

    if (currX == NULL || currY == NULL)
        return;

    if (prevX != NULL)
        prevX->next = currY;
    else
        lista->head = currY;

    if (prevY != NULL)
        prevY->next = currX;
    else
        lista->head = currX;
    Node * temp = currX->next;
    currX->next = currY->next;
    currY->next = temp;
}

void Swap(){
    printf("dati Id-urile farmaciilor care urmeaza a fi interschimbate:\n\tb1 = ");
    int b1, b2;
    scanf("%d", &b1);
    printf("\tb2 = ");
    scanf("%d", &b2);
    swapNodes(b1,b2);
    setIndexes(lista);
    printf("Elementele au fost interschimbate cu succes\n");
}

void AddLast(){
    Farmacie * c = ReadStruct();
    if (c==NULL)
    {
        printf("Eroare la alocarea memoriei\n");
        return;
    }
    if (lista==NULL)
    {
        lista = new_list();
    }
    push_last(lista, c);
    printf("Farmacia a fost adaugata cu succes\n");
}
```

```
void AddFirst(){
    Farmacie * c = ReadStruct();
    if (c==NULL)
    {
        printf("Eroare la alocarea memoriei\n");
        return;
    }
    if (lista==NULL)
    {
        lista = new_list();
    }
    push_first(lista, c);
    printf("Farmacia a fost adaugata cu succes\n");
}

void Delete(){
    printf("Introduceti ID-ul structurii ce urmeaza a fi stearsa: ");
    int ID = 0;
    scanf("%d", &ID);
    if (ID>lista->Count-1)
    {
        printf("Ivalid index\n");
        return;
    }
    deleteNthNode(lista, ID);
    printf("Farmacia a fost stearsa cu succes\n");
}

void AddAfter(){
    printf("Introduceti ID-ul elementului dupa care urmeaza a fi adaugat un nou
element: ");
    int ID = 0;
    scanf("%d", &ID);
    if (ID>lista->Count-1)
    {
        printf("Ivalid index\n");
        return;
    }
    Node * curent = lista->head;
    while (curent!=NULL)
    {
        if (ID==curent->index)
        {
            break;
        }
        curent = curent->next;
    }
    Farmacie * c = ReadStruct();
    if (c==NULL)
    {
        printf("Eroare la alocarea memoriei\n");
        return;
    }
    insertAfterNthNode(ID+1, c, lista);
    printf("Farmacia a fost adaugata cu succes\n");}
```

```
void AddBefore(){
    printf("Introduceti ID-ul elementului inaintea caruia urmeaza a fi adaugat un nou
element: ");
    int ID = 0;
    scanf("%d", &ID);
    if (ID>lista->Count-1)
    {
        printf("Ivalid index\n");
        return;
    }
    Node * curent = lista->head;
    while (curent!=NULL)
    {
        if (ID==curent->index)
        {
            break;
        }
        curent = curent->next;
    }
    Farmacie * c = ReadStruct();
    if (c==NULL)
    {
        printf("Eroare la alocarea memoriei\n");
        return;
    }
    insertAfterNthNode(ID-1,c, lista);
    printf("Farmacia a fost adaugata cu succes\n");
}

void Export(){
    if (lista==NULL)
    {
        printf("Lista nula\n");
        return;
    }
    if (is_empty(lista))
    {
        printf("Lista goala");
        return;
    }
    FILE * f = fopen("lista.txt", "w");
    if (f==NULL)
    {
        printf("Eroare la deschiderea fisierului\n");
        return;
    }
    Node * curent_node = lista->head;
    while (curent_node!=NULL)
    {
        fprintf(f, "%s", ToString(curent_node->value));
        curent_node = curent_node->next;
    }
    fclose(f);
    printf("Datele au fost exportate cu succes\n");
}
```

```
void Import(){
    FILE * f = fopen("lista.txt", "r");
    if (f==NULL)
    {
        printf("Eroare la deschiderea fisierului\n");
        return;
    }
    char * str = (char *)malloc(1000);
    while (fgets(str, 1000, f)!=NULL)
    {
        Farmacie * c = FromString(str);
        push_first(lista, c);
    }
    fclose(f);
    printf("Datele au fost importate cu succes\n");
}


int userChose = 0;
int go = 1;

int Menu(){
    printf("-------------------------Meniul Aplicatiei-------------------------\n");
    printf("1 . \tCrearea listei in memoria dinamica\n");
    printf("2 . \tIntroducerea informatiei despre elementele listei de
latastatura\n");
    printf("3 . \tAfiseaza informatia despre elementele listei la ecran\n");
    printf("4 . \tCautarea elementului in lista\n");
    printf("5 . \tModificarea campurilor unui element din lista\n");
    printf("6 . \tDeterminarea adresei ultimului element din lista\n");
    printf("7 . \tDeterminarea lungimii listei\n");
    printf("8 . \tInterschimbarea a doua elemente indicate in lista\n");
    printf("9 . \tSortarea listei\n");
    printf("10 . \tAdauga element la inceputul listei\n");
    printf("11 . \tAdauga element dupa un element indicat\n");
    printf("12 . \tAdauga element inaintea unui element idicat\n");
    printf("13 . \tStergerea elementului indicat in lista\n");
    printf("14 . \tExport informatii in fisier\n");
    printf("15 . \tImport informatii in fisier\n");
    printf("16 . \tEliberarea memoriei alocate pentru lista\n");
    printf("0 . \tExit\n");
    printf("-----------------------------------------------------------------
\nOptiunea aleasa --> ");
    int op;
    scanf("%d", &op);
    system("cls");
    return op;
}

void PressAnyKey(){
    printf("\nAtingeti o tasta pentru a continua\n");
    getch();
    system("cls");
}
```

```
void BL(){
    switch (userChose)
    {
        case 1:
        {
            lista = new_list();
            printf("Lista a fost creata cu succes");
            PressAnyKey();
        } break;
        case 2:
        {
            Read();
            PressAnyKey();
        }break;
        case 3:
        {
            ShowList();
            PressAnyKey();
        }break;
        case 4:
        {
            Search();
            PressAnyKey();
        }break;
        case 5:
        {
            Update();
            PressAnyKey();
        }break;
        case 6:
        {
            LastIndex();
            PressAnyKey();
        }break;
        case 7:
        {
            printf("Lista contine %d elemente", lista->Count);
            PressAnyKey();
        }break;
        case 8:
        {
            Swap();
            PressAnyKey();
        }break;
        case 9:
        {
            SortList();
            PressAnyKey();
        }break;
        case 10:
        {
            AddFirst();
            PressAnyKey();
        }break;
        case 11:
```

```
        {
            AddAfter();
            PressAnyKey();
        }break;
        case 12:
        {
            AddBefore();
            PressAnyKey();
        }break;
        case 13:
        {
            Delete();
            PressAnyKey();
        }break;
        case 14:
        {
            Export();
            PressAnyKey();
        }break;
        case 15:
        {
            Import();
            PressAnyKey();
        }break;
        case 16:
        {
            clear_list(lista);
            is_empty(lista)?
                printf("succes\n"):
                printf("error\n");
            delete_list(lista);
            PressAnyKey();
        }break;
        case 0:
        {
            go=0;
            system("cls");
            printf("Aplicatia s-a oprit cu succes");
            getch();
        }break;
        default:printf("Optiune necunoscuta\nIncercati din nou");break;
    }
}
```

list.h

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct Node{
    int index;
    void * value;
    struct Node * next;
}Node;

typedef struct List{
```

```
    int Count;
    Node * head;
}List;

List * new_list(){
    List * list = (List*)malloc(sizeof(List));
    list->head = NULL;
    list->Count=0;
    return list;
}

Node * new_node(void * data){
    Node * node = (Node *)malloc(sizeof(Node));
    node->value = data;
    node->next = NULL;
    return node;
}
int setIndexes(List *list){
    int n = 0;
    Node * curent_node = list->head;
    while (curent_node!=NULL)
    {
        curent_node->index = n;
        n++;
        curent_node = curent_node->next;
    }
    return n;
}

bool push_first(List *list, void * data){
    Node * newNode = new_node(data);
    newNode->next = list->head;
    list->head = newNode;
    list->Count++;
    int n = setIndexes(list);
    if (n == list->Count)
    {
        return true;
    }
    return false;
}

bool push_last(List *list, void * data){
    struct Node *newNode = new_node(data);
    newNode->value = data;
    newNode->next = NULL;
    if(list->head == NULL)
            list = new_list();
    else
    {
        struct Node *lastNode = list->head;
        while(lastNode->next != NULL)
        {
            lastNode = lastNode->next;
        }
```

```
        lastNode->next = newNode;
    }
    list->Count++;
    int n = setIndexes(list);
    if (n == list->Count)
    {
        return true;
    }
    return false;
}

void * pop_first(List *list){
    Node * old_node = list->head;
    void * return_value = 0;
    if(old_node)
    {
        list->head = list->head->next;
        return_value = old_node->value;
        free(old_node);
    }
    list->Count--;
    return return_value;
}

void clear_list(List *list){
    Node * current_node = list->head, * back_node =NULL;
    while(current_node)
    {
        back_node = current_node;
        current_node=current_node->next;
        free(back_node);
    }
    list->head = NULL;
    list->Count = 0;
}

void delete_list(List *list){
    clear_list(list);
    free(list);
}

bool is_empty(List *a){
    return (a->head)?false:true;
}

void tailRecRevese(Node* current, Node* previous, Node** head){
    if (!current->next)
    {
        *head = current;
        current->next = previous;
        return;
    }
    Node* next = current->next;
    current->next = previous;
    tailRecRevese(next, current, head);}
```

```
void tailRecReveseLL(Node** head){
    if (!head)
        return;
    tailRecRevese(*head, NULL, head);
}

void reverse(List * a){
    tailRecReveseLL(&(a->head));
    printf("Lista a fost inversata\n");
}

void deleteNthNode(List * lista, int n){
    if(n < 0 || n >= lista->Count)
        printf("pozitie de stergere invalida\n");
    else if(n == 0)
    {
        Node * temp = lista->head;
        lista->head = lista->head->next;
        free(temp);
        lista->Count--;
    }
    else
    {
        Node * temp = lista->head;
        while(--n)
        {
            temp=temp->next;
        }
        Node * temp2 = temp->next;
        temp->next = temp2->next;
        free(temp2);
        lista->Count--;
    }
    int x = setIndexes(lista);
    if (x == lista->Count)
    {
        printf("Nodul a fost sters\n");
    }
}

bool insertAfterNthNode(int n, void * data, List * lista){
    struct Node* newNode = new_node(data);
    if(n < 0 || n > lista->Count)
        printf("pozitie de inserare invalida\n");
    else if(n == 0)
    {
        newNode->next = lista->head;
        lista->head = newNode;
        lista->Count++;
    }
    else
    {
        struct Node* temp = lista->head;
        while(--n)
        {
```

```
            temp=temp->next;
        }
        newNode->next= temp->next;
        temp->next = newNode;
        lista->Count++;
    }
    int x = setIndexes(lista);
    if (x == lista->Count)
    {
        return true;
    }
    return false;
}
```

4. Testarea aplicației

```
------------------------Meniul Aplicatiei------------------------
1 .     Crearea listei in memoria dinamica
2 .     Introducerea informatiei despre elementele listei de latastatura
3 .     Afiseaza informatia despre elementele listei la ecran
4 .     Cautarea elementului in lista
5 .     Modificarea campurilor unui element din lista
7 .     Determinarea lungimii listei
8 .     Interschimbarea a doua elemente indicate in lista
9 .     Sortarea listei
10 .    Adauga element la inceputul listei
11 .    Adauga element dupa un element indicat
12 .    Adauga element inaintea unui element idicat
13 .    Stergerea elementului indicat in lista
14 .    Export informatii in fisier
15 .    Import informatii in fisier
16 .    Eliberarea memoriei alocate pentru lista
0 .     Exit
------------------------------------------------------------------
Optiunea aleasa -->
```

Nr.	Input	Output
1.	Introducerea informației despre elementele listei de la tastatura	Dati datele Farmaciei: ========== Denumire: Felicia Adresa: 1 Telefon: 1 Ore de lucru: 1 Volum medicamente: 1 succes Mai introduceti Farmacii(y/n)? : y Dati datele Farmaciei: ========== Denumire: Hipocrate Adresa: ads Telefon: ads Ore de lucru: ads Volum medicamente: 123 succes Mai introduceti Farmacii(y/n)? : y

2.	Afișarea informațiilor despre elementele listei	``` ========== Farmacie 0: Denumire: VitaFamily Adresa: ads Telefon: asd Ore de lucru: asd Volum medicamente: 213 ========== ========== Farmacie 1: Denumire: Sancons Adresa: ads Telefon: asd Ore de lucru: asd Volum medicamente: 213 ========== ========== Farmacie 2: Denumire: Familia Adresa: ads Telefon: asd Ore de lucru: asd Volum medicamente: 213 ========== ========== Farmacie 3: Denumire: Hipocrate Adresa: ads Telefon: asd Ore de lucru: asd Volum medicamente: 213 ========== ```
3.	Căutarea unui element în listă	``` Introduceti Numele farmaciei:Familia ========== Farmacie 2: Denumire: Familia Adresa: ads Telefon: asd Ore de lucru: asd Volum medicamente: 213 ========== Introduceti Numele farmaciei:x Elementul cautat nu se afla in lista ```

4.	Modificarea câmpurilor unui element în listă	Introduceti ID-ul Farnaciei ce urmeaza a fi actualizata: 4 Modificati denumirea farmaciei?(y/n) ->n Modificati adresa farmaciei?(y/n) ->n Modificati telefonul farmaciei?(y/n) ->y Telefonul farmaciei: 069329255 Modificati orele de lucru?(y/n) ->n Modificati volumul de medicamente?(y/n) ->y Volumul de medicamente: 200 Datele datele au fost actualziate ========== Farmacie 4: Denumire: Felicia Adresa: 1 Telefon: 069329255 Ore de lucru: 1 Volum medicamente: 200 ==========
5.	Determinarea adresei ultimului element din listă și lungimii listei	Lista contine 5 elemente Atingeti o tasta pentru a continua Ultimul nod se afla pe pozitia 4 si are adresa in memorie: 0061FEEC Atingeti o tasta pentru a continua
6.	Interschimbarea a 2 elemente	dati Id-urile farmaciilor care urmeaza a fi interschimbate: b1 = 1 b2 = 2 Elementele au fost interschimbate cu succes
7.	Adăugarea unui element	Dati datele Farmaciei: ========== Denumire: HelperMed Adresa: str Korolenco 9 Telefon: 123456789 Ore de lucru: x Volum medicamente: 200 Farmacia a fost adaugata cu succes
8.	Ștergerea elementului indicat	Introduceti ID-ul structurii ce urmeaza a fi stearsa: 3 Nodul a fost sters Farmacia a fost stearsa cu succes
9.	Exportarea informațiilor în fișier	Datele au fost importate cu succes lista.txt 1 Felicia_|_1_|_1_|_1_|_1 2 Hipocrate_|_ads_|_asd_|_asd_|_213 3 Familia_|_ads_|_asd_|_asd_|_213 4 Sancons_|_ads_|_asd_|_asd_|_213 5 VitaFamily_|_ads_|_asd_|_asd_|_213
10.	Importarea informațiilor din fișier	Datele au fost importate cu succes

5. Concluzii

În concluzie, lucrarea de laborator realizată a reprezentat o oportunitate de aplicare a cunoștințelor teoretice în practică. Lucrarea de laborator a avut ca obiectiv principal familiarizarea și implementarea practică a tipului abstract de date "Listă simplu înlănțuită" în limbajul C, cu accent pe operațiile de prelucrare de bază ale listei. Prin intermediul celor trei fișiere-text create conform cerințelor specificate, am reușit să dezvolt o structură eficientă pentru gestionarea datelor într-o farmacie, inclusiv funcționalități precum crearea și gestionarea listei de medicamente, introducerea și afișarea informațiilor, căutarea, modificarea, sortarea și eliberarea memoriei alocate. Acestea oferă un instrument versatil pentru manipularea datelor într-un context specific, contribuind la înțelegerea și aplicarea practică a conceptelor legate de structurile de date în programarea în limbajul C.