Structuri de Date și Algoritmi
Lucrarea de laborator nr. 5

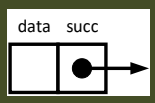
17 martie 2020

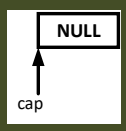
ș.l.dr.ing. Corina Cîmpanu

Cuprins

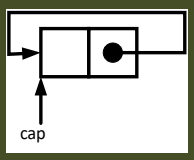
Liste circulare simplu înlănțuite
Structura

```
struct Elem {
    int data;
    Elem* succ;
};
```

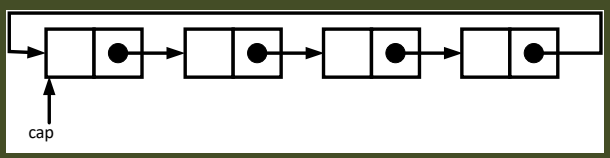




Listă vidă



Listă cu un singur element



Listă cu mai multe elemente

Liste circulare simplu înlănțuite
Operații

1.1. Inițializarea
1.2. Inserarea în față
1.3. Afișarea
1.4. Inserarea unei valori pe o poziție
1.5. Ștergerea unui element de pe o poziție dată
1.6. Eliberarea memoriei

Main.cpp | Functii.cpp | Header.h*

LCSI | (Global Scope)

```
#pragma once

#include <iostream>
using namespace std;

struct Elem {
    int data;
    Elem* succ;
};

void Initializare(Elem* &cap);
void InserareFata(Elem* &cap, int val);
void Afisare(Elem* cap);
void InserarePozitie(Elem* &cap, int val, int poz);
void StergePozitie(Elem* &cap, int poz);
void EliberareMemorie(Elem* &cap);
```

Liste circulare simplu înlănțuite
1.1. Inițializarea

Main.cpp | Functii.cpp | Header.h*

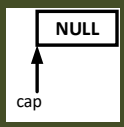
LCSI | (Global Scope)

```
#include "Header.h"

void Initializare(Elem* &cap)
{
    cout << "\n Initializarea listei...";
    cap = 0;
}
```

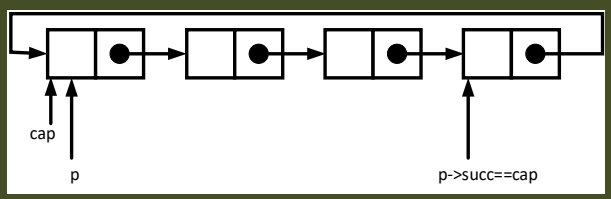
Main.cpp | Functii.cpp | Header.h

LCSI | (Globa

```
#include "Header.h"

int main()
{
    Elem* cap;
    int n = 10;

    Initializare(cap);
    Afisare(cap);
```

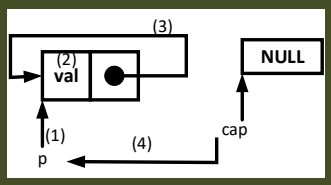


Liste circulare simplu înlănțuite
1.2. Afișarea



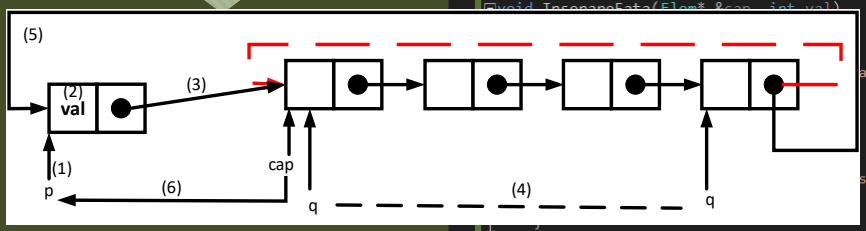

```
void Afisare(Elem* cap)
{
    Elem *p = cap;
    cout << "\n Afisarea elementelor listei ...\n";
    if (p)
    {
        while (p->succ != cap)
        {
            cout << p->data << " ";
            p = p->succ;
        }
        cout << p->data << " ";
    }
    else
        cout << "\n Lista vida. \n";
}
```

Liste circulare simplu înlănțuite
1.3. Inserarea în fața listei (1)



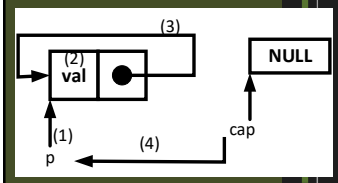

```
void InsereareFata(Elem* &cap, int val)
{
    Elem* p, *q;
    cout << "\n Inserare in fata listei a valorii val = " << val << " ...";
    p = new Elem;
    p->data = val;
    if (cap == 0)
    {
        cout << "\n Cazul lista vida, inserez primul element ...";
        p->succ = p;
    }
    else
    {
        cout << "\n Cazul in care lista are deja elemente ...";
        p->succ = cap;
        q = cap;
        while (q->succ != cap)
            q = q->succ;
        q->succ = p;
    }
    cap = p;
}
```

Liste circulare simplu înlănțuite
1.3. Inserarea în fața listei (2)

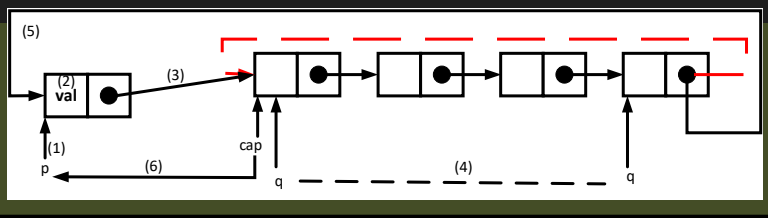



```
    else
    {
        cout << "\n Cazul in care lista are deja elemente ...";
        p->succ = cap;
        q = cap;
        while (q->succ != cap)
            q = q->succ;
        q->succ = p;
    }
    cap = p;
}
```

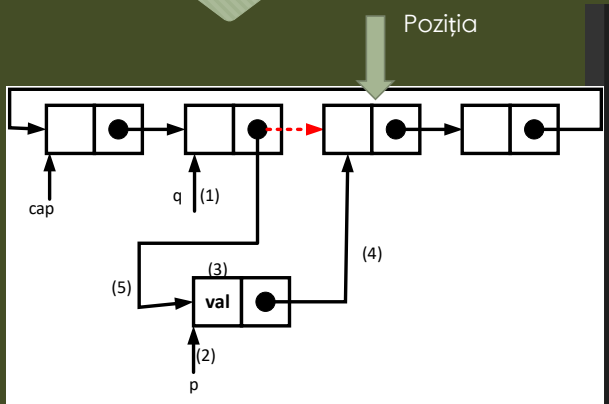
Liste circulare simplu înlănțuite
1.4. Inserarea pe o poziție dată




```
void InserarePozitie(Elem* &cap, int val, int poz)
{
    Elem *p, *q;
    int contor;

    cout << "\n Inserare poztie intermediara poz = " << poz << " a valorii val = " << val << " ...";
    if (poz <= 0)
        cout << "\n Pozitie invalida - prea mica ...";
    else
    {
        if (poz == 1)
            InserareFata(cap, val);
        else
        {
```

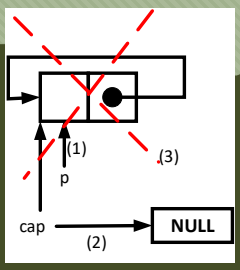


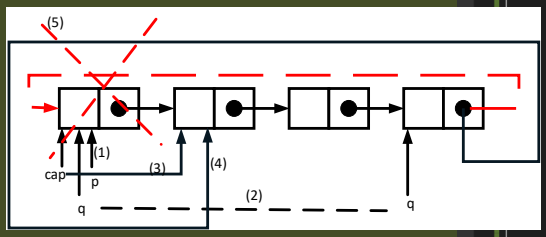
Liste circulare simplu înlănțuite
1.4. Inserarea pe o poziție dată (2)




```
        else
        {
            contor = 1;
            q = cap;
            while (contor < poz - 1 && q->succ != cap)
            {
                contor++;
                q = q->succ;
            }
            if (contor == poz - 1)
            {
                cout << "\n Pozitie intermediara ...";
                p = new Elem;
                p->data = val;
                p->succ = q->succ;
                q->succ = p;
            }
            else
                cout << "\n Pozitie invalida - prea mare ...";
        }
    }
}
```

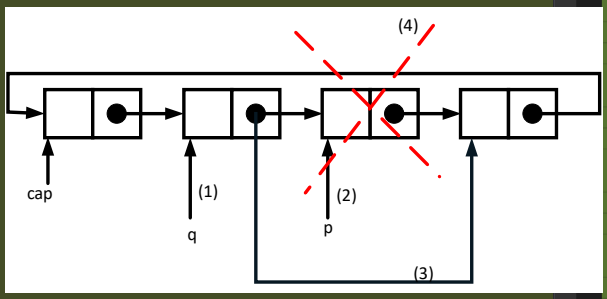
Liste circulare simplu înlănțuite
1.5. Ștergerea de pe o anumită poziție (1)






```
void StergePozitie(Elem* &cap, int poz)
{
    Elem *p, *q;
    int contor;

    cout << "\n Stergere poztie intermediara poz = " << poz << " ...";
    if (poz <= 0)
        cout << "\n Pozitie invalida - prea mica ...";
    else
    {
        if (poz == 1)
        {
            cout << "\n Stergerea capului listei ...";
            p = cap;
            if (cap->succ == cap)
            {
                cout << "\n Stergerea singurului element din lista ...";
                cap = 0;
            }
            else
            {
                cout << "\n Stergerea capului listei cand lista are mai mult de un element...";
                q = cap;
                while (q->succ != cap)
                    q = q->succ;
                cap = cap->succ;
                q->succ = cap;
            }
            delete p;
            p = 0;
```

Liste circulare simplu înlănțuite
1.5. Ștergerea de pe o anumită poziție (2)




```
        else
        {
            contor = 1;
            q = cap;
            while (contor < poz - 1 && q->succ != cap)
            {
                contor++;
                q = q->succ;
            }
            if (contor == poz - 1)
            {
                cout << "\n Pozitie intermediara ...";
                p = q->succ;
                q->succ = p->succ;
                delete p;
                p = 0;
            }
            else
                cout << "\n Pozitie invalida - prea mare ...";
        }
    }
}
```

Liste circulare simplu înlănțuite
1.6. Eliberarea memoriei

```
void EliberareMemorie(Elem* &cap)
{
    cout << "\n Eliberarea memoriei ...";
    while (cap)
    {
        StergePozitie(cap, 1);
    }
    cap = 0;
}
```

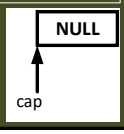
Liste circulare simplu înlănțuite
Verificare

```
    for (int i = 0;i < n;i++)
        InserareFata(cap, n - i);

    Afisare(cap);

    InserarePozitie(cap, 101, -20);
    InserarePozitie(cap, 201, 0);
    InserarePozitie(cap, 301, 1);
    InserarePozitie(cap, 401, 11);
    InserarePozitie(cap, 501, 5);
    InserarePozitie(cap, 601, 21);
    Afisare(cap);

    StergePozitie(cap, -30);
    StergePozitie(cap, 0);
    StergePozitie(cap, 1);
    StergePozitie(cap, 10);
    StergePozitie(cap, 5);
    StergePozitie(cap, 200);
    Afisare(cap);

    EliberareMemorie(cap);
    Afisare(cap);

    system("PAUSE");
    return 0;
}
```

Liste dublu înlănțuite
Structura, operațiile și inițializarea

Operatii

1.1. Inițializarea
1.2. Inserarea în față
1.3. Afișarea
1.4. Afișarea dus - întors
1.5. Inserarea unei valori pe o poziție
1.6. Ștergerea unui element de pe o poziție dată
1.7. Eliberarea memoriei

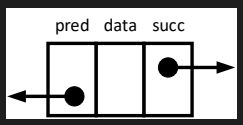


Inițializarea

Main.cpp | Functii.cpp* | Header.h*

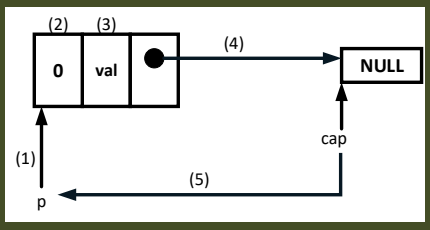
LDI | (Global Scope)

```
#pragma once

#include <iostream>
using namespace std;

struct Elem {
    Elem* pred;
    int data;
    Elem* succ;
};


void Initializare(Elem* &cap);
void InserareFata(Elem* &cap, int val);
void Afisare(Elem* cap);
void AfisareDusIntors(Elem* cap);
void InserarePozitie(Elem* &cap, int val, int poz);
void StergerePozitie(Elem* &cap, int poz);
void EliberareMemorie(Elem* &cap);
```



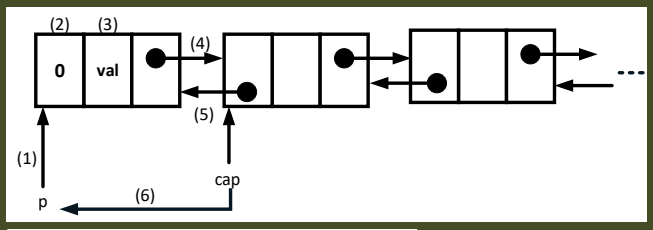
Liste dublu înlănțuite
1.2. Inserarea în față

Cazul 1. Inserarea în față într-o listă vidă



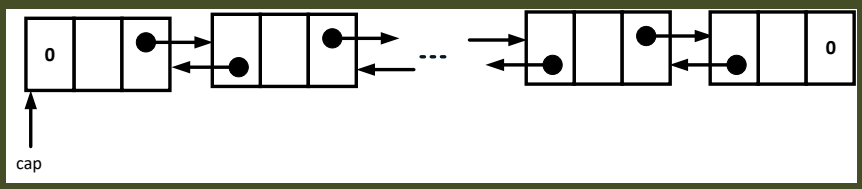
(1)
(2)
(3)
(4)
(5)

Cazul 2. Inserarea în față într-o listă în care există deja elemente

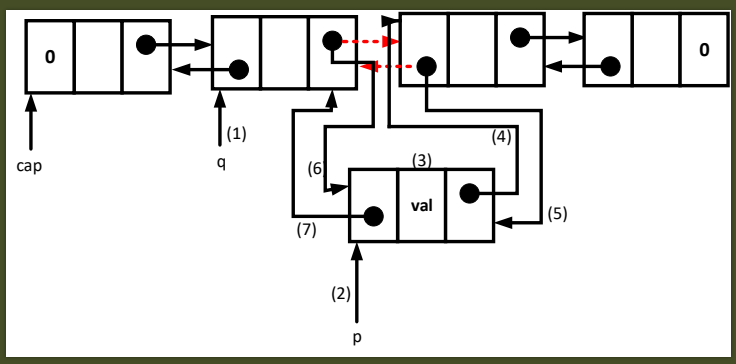


(1)
(2)
(3)
(4)
(5)
(6)

Liste dublu înlănțuite
1.3. Afișarea simplă și
1.4. Afișarea dus-întors

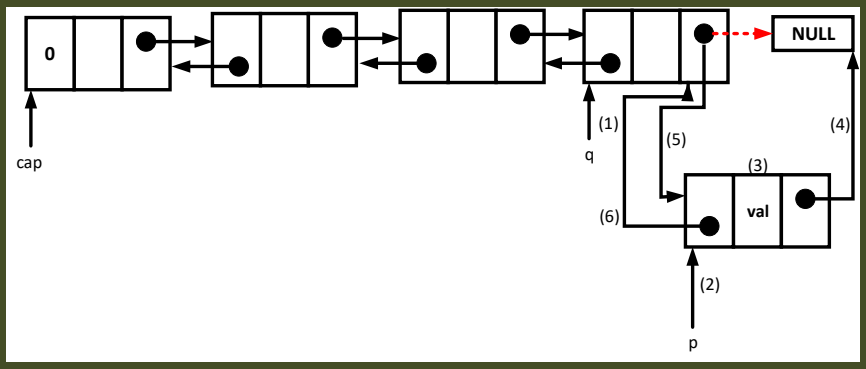


Liste dublu înlănțuite
1.5. Inserarea pe o anumită poziție (1) intermediară



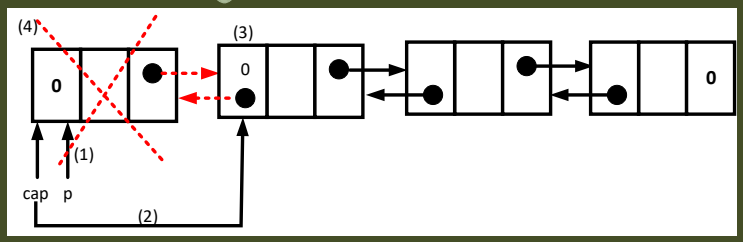
(1)
(2)
(3)
(4)
(5)
(6)
(7)

Liste dublu înlănțuite
1.5. Inserarea pe o anumită poziție (2) ultima



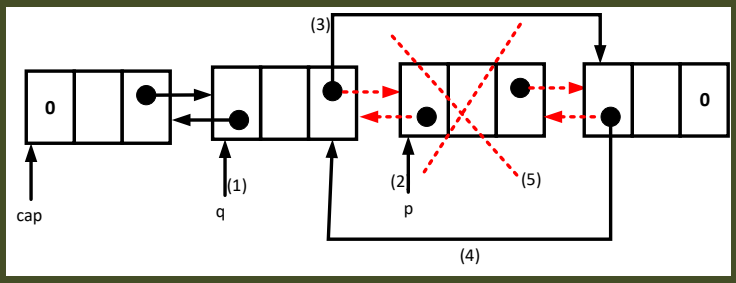
(1)
(2)
(3)
(4)
(5)
(6)

Liste dublu înlănțuite
1.6. Ștergerea de pe o anumită poziție (1) primului element



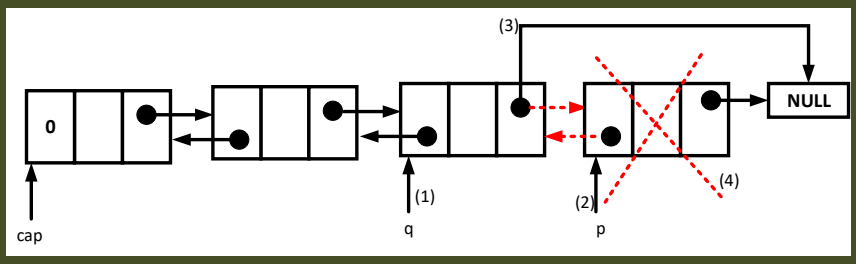
(1)
(2)
(3)
(4)

Liste dublu înlănțuite
1.6. Ștergerea de pe o anumită poziție (2) intermediar



(1)
(2)
(3)
(4)
(5)

Liste dublu înlănțuite
1.6. Ștergerea de pe o anumită poziție (3) ultimul



(1)
(2)
(3)
(4)

1.7. Eliberarea memoriei

- Se va șterge pe rând elementul de pe prima poziție cât timp există elemente în listă

Probleme propuse

1. Lista circulară simplu înlănțuită – implementarea operațiilor de bază (pașii pe hârtie)
2. Lista circulară simplu înlănțuită – implementarea operațiilor de bază (cod)
3. Lista dublu înlănțuită – implementarea operațiilor de bază (pașii (1)….(n) pe hârtie)
4. Lista dublu înlănțuită – implementarea operațiilor de bază (cod)
5. Concatenare LCSI
6. Concatenare LDI
7. Interclasare LCSI
8. Interclasare LDI
9. Inversare legături LCSI
10. Joseph – ordinea persoanelor executate
11. Joseph – persoana salvată
12. Lista circulară dublu înlănțuită – operații de bază pe hârtie
13. LCDI – implementarea operațiilor de bază (cod)
14. LCDI – reprezentarea numerelor mari