MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII AL REPUBLICII MOLDOVA

IP CENTRUL DE EXCELENȚĂ ÎN INFORMATICĂ ȘI TEHNOLOGII INFORMAȚIONALE

CATEDRA INFORMATICA II

ANDRIAN DASCAL

PROGRAMAREA CALCULATORULUI

STRUCTURI DINAMICE DE DATE ÎN C++

Îndrumar pentru activitățile practice,
destinat cadrelor didactice și elevilor din IP CEITI

CHIȘINĂU, 2020

Aprobat: *Consiliul metodic-ştiinţific, IP CEITI, proces verbal Nr. 8, din data 30.03.21, _____*

_____

Elaborat conform Curriculumului modular „Programarea calculatorului”, ediția 2020. Discutat și examinat la ședința catedrei „Informatica II” din 12.11.2020, Proces Verbal Nr. 2, șef catedră, Luminița Gribineț _____

Autor: *Andrian Dascal, magistru în Informatică şi Tehnologii Informaționale, grad didactic II, IP CEITI.*

Recenzie: *Ioan Jeleascov, prof. la discipline de informatică, IP CEITI.*

Redactare ştiinţifică: *Ioan Jeleascov, „architect advisor” în cadrul unității "StoneHard".*

Redactare lingvistică: *Doina Lupu, prof. de limba şi literatura română, gr. didactic I, IP CEITI.*

DRAGI PRIETENI,

„Educația în domeniul informaticii nu poate face pe nimeni un programator expert, decât să studieze pensulele și pigmentul care-l pot face pe cineva un pictor expert”.

Eric S. Raymond

Cunoaștem că C++ este un limbaj de programare creat de Bjarne Stroustrup ca o extensie a limbajului de programare C sau „C cu clase”. Limbajul C++ s-a extins semnificativ de-a lungul timpului, iar actualul limbaj C++ este modern, cu funcții orientate spre obiecte, cu funcții generice și funcționale. Pe lângă facilități pentru manipularea memoriei la nivel scăzut, un rol important în susținerea programatorilor este implementarea bibliotecilor de șabloane standard (BȘS).

BȘS este un set de clase de șabloane în C++ pentru a furniza structuri și funcții comune de date, cum ar fi: liste, stive, masive etc. Este o bibliotecă de clase de: containere, algoritmi și iteratori. BȘS este generalizată, astfel componentele sale sunt parametrizate. O cunoaștere de lucru a claselor de șabloane este o condiție prealabilă pentru lucrul cu BȘS.

Lucrarea “STRUCTURI DINAMICE DE DATE ÎN C++” reprezintă un suport teoretic și praxiologic destinat primei unități de conținut din unitatea de curs “PROGRAMAREA CALCULATORULUI”. Lucrarea are drept scop însușirea de către elevi a cunoștințelor necesare dezvoltării gândirii algoritmice și formării culturii informaționale.

Materialul inclus are menirea de a dezvoltarea următoarelor competențe: analiza structurală a problemei; divizarea problemelor complexe în probleme mai simple şi reducerea lor la cele deja rezolvabile; utilizarea metodelor formale pentru elaborarea algoritmilor şi scrierea programelor respective în limbajul de programare C/C++. Realizarea acestui obiectiv presupune extinderea capacităţilor fiecărui elev în elaborarea algoritmilor destinați rezolvării problemelor cu implementarea structurilor dinamice de date întâlnite în activitatea cotidiană a unui programator.

Acest îndrumar este destinat cadrelor didactice și elevilor din IP. CEITI, fiecărei persoane care are dorință de a studia domeniul programării la un nivel mai superior. Mult succes la fiecare!

Autorul

CUPRINSUL

4. LISTE CIRCULARE SIMPLU ŞI DUBLU ÎNLĂNŢUITE

5. STRUCTURA DINAMICĂ STIVA

6. STRUCTURA DINAMICĂ COADA

7. STRUCTURA DINAMICĂ ARBORELE

INTRODUCERE

1. Motivaţia, utilitatea modulului pentru dezvoltarea profesională

Studierea acestui modul contribuie la formarea şi dezvoltarea competenţelor profesionale ce corespund nivelului patru de calificare:

- *cunoștințe faptice, principii, procese și concepte generale din domeniul elaborării produselor program;*
- *abilități cognitive și practice necesare pentru elaborarea aplicațiilor de consolă conform tematicilor incluse;*
- *asumarea responsabilității pentru mentenanța de aplicații.*

Competenţele formate şi dezvoltate în cadrul acestui modul sunt necesare studierii unităților de curs orientate spre elaborarea și dezvoltarea produselor program. Aceste competențe sunt de o reală importanță în activitatea profesională a tehnicianului, în special, în domeniul gestiunii produselor-program utilizate în companii.

2. Competenţele profesionale specifice modulului

Modulul respectiv permite formarea și dezvoltarea următoarelor competențe profesionale specifice:

- *Prelucrarea tipurilor dinamice de date în cadrul aplicațiilor de consolă.*
- *Utilizarea structurilor dinamice de date pentru problemele întâlnite în activitatea profesională.*

3. Repartizarea orientativă a orelor pe unitatea de învăţare

Unitatea de învățate	**Numărul de ore**			
	Contact direct		*Activitate individuală*	**Total**
	Teorie	*Practică*		
Structuri dinamice de date	**12**	**12**	**12**	**36**

4. Abilități profesionale specifice unității de învățare

Studierea componentelor modulului formează şi dezvoltă următoarele abilități profesionale specifice:

- *declararea unei structuri dinamice de date;*
- *alocarea dinamică a memoriei unei variabile dinamice;*
- *eliberarea memoriei dinamice alocate unei variabile dinamice;*
- *crearea unei structuri dinamice de date de tip listă, stivă, coadă, arbore binar conform specificațiilor propuse;*
- *afișarea, căutarea, permutarea, eliminarea, adăugarea, ordonarea datelor unei structuri dinamice de date de tip listă, stivă, coadă, arbore binar conform specificațiilor propuse;*
- *interclasarea și distrugerea structurilor dinamice de date de tip listă, stivă, coadă, arbore binarconform specificațiilor propuse;*
- *elaborarea algoritmilor pentru tipuri de date de tip listă, stivă, coadă, arbore binar;*
- *translarea și implementarea algoritmilor pentru tipuri de date de tip listă, stivă, coadă, arbore binar în limbajul de programare.*

5. Studiul individual ghidat de profesor[1]

Subiecte	Produse de elaborat	Evaluarea
Liste liniare și circulare	*Set de aplicații de consolă ce aplică structurile dinamice liniare (listele liniare simplu înlănțuite și dublu înlănțuite) și cele circulare (listele circulare simplu înlănțuite și dublu înlănțuite)*	*Prezentarea portofoliului în format electronic*
Stive și cozi	*Set de aplicații de consolă ce aplică structurile dinamice de tip stivă și coadă.*	*Prezentarea portofoliului în format electronic*
Arbori binari	*Set de aplicații de consolă ce aplică structurile dinamice arborescente.*	*Prezentarea portofoliului în format electronic*

1 *Curriculum modular „**Programarea calculatorului**”, ediția 2016 și 2020.*

1. PERSPECTIVA C++ STL

1.1 Prezentare generală pentru C++ STL

În programarea calculatoarelor, șabloanele sunt o caracteristică a limbajului de programare C++ ce permit scrierea de cod fără a lua în considerare tipul de dată ce va fi utilizat. Șabloanele permit programarea generică în limbajul C++, fiind utile programatorilor, mai ales când sunt combinate cu tehnica moștenirilor multiple și a supraîncărcării operatorilor. BȘS a limbajului C++ aduce multe funcții utile într-un cadru de șabloane conectate.

C++ STL (The Standard Template Library) este o mulțime puternică de clase de șabloane C++, notată anterior BȘS[2]. La temelia C++ STL exisă următoarele trei componente:

Nr.	*Componenta*	*Descrierea*
1	*Containere*	*Sunt utilizate pentru gestionarea colecțiilor de obiecte de un anumit tip. Există mai multe tipuri diferite de containere, cum ar fi: deque, list, vector, map etc.*
2	*Iteratori*	*Sunt folosiți pentru a parcurge elementele colecțiilor de obiecte. Aceste colecții pot fi containere sau subseturi de containere.*
3	*Algoritmi*	*Acționează asupra containerelor. Acestea oferă unele mijloace prin care putem efectua inițializarea, sortarea, căutarea și transformarea conținuturilor containerelor.*

C++ STL încorporează, de asemenea, toate bibliotecile Standard C, cu mici modificări pentru a susține siguranța tipului.

BȘS în C++ pot fi clasificate în două ramuri:

1. ***Biblioteca de funcții standard (BFS)** - constă din funcții de uz general, independente, care nu fac parte din nicio clasă. Biblioteca de funcții este moștenită de la limbajul C.*

2 *Biblioteca de șabloane standard (BȘS) în limbajul C++, denumirea a fost tradusă din expresia engleză „**Standard Template Library**”, notată prin **STL**.*

2. ***Biblioteca de clase orientată pe obiecte (BCOO)*** *- aceasta este o colecție de clase și funcții asociate*[3].

1.2 Avantaje şi dezavantaje pentru C++ STL

Unele aplicații ale șabloanelor, cum ar fi funcția ***max()****, au fost mai devreme suplinite de către preprocesor prin intermediul macrourilor. De exemplu, iată un cod macro pentru* ***max()****:*

```
#define max(a,b)    ((a) < (b) ? (b) : (a))
```

Remarcă:

> *Un macro este un fragment de cod căruia i s-a dat un nume. Ori de câte ori este folosit numele, acesta este înlocuit cu conținutul macro-ului.*

Există două tipuri de macro-uri, acestea diferă mai ales în ceea ce aratăcând sunt aplicate. Putem defini orice identificator valid ca macro, chiar dacă este un cuvânt cheie C.

Preprocesatorul nu știe nimic despre cuvintele cheie. Acest lucru poate fi util dacă dorim să ascundem un cuvânt cheie, cum ar fi const, dintr-un compilator mai vechi care nu îl înțelege. Operatorul de preprocesor definit nu poate fi definit niciodată ca o macro și operatorii numiți de C++ nu pot fi macrocomenzi atunci când compilăm un cod în limbajul C++.

Codul unui macro este întotdeauna inserat în program în locul unde apare un apel către acesta. Șabloanele sunt tratate ca funcții, deși compilatorul poate decide să insereze pe loc cod în loc de un apel de funcție, dacă consideră acest lucru oportun. Macro-instrucțiunile și șabloanele-funcție nu constituie o corvoadă (activitate în plus) la momentrul execuției.

Șabloanele sunt în general considerate ca fiind o îmbunătățire a macrourilor, acestea sunt sigure din punctul de vedere al coerenței tipului de date utilizate. Șabloanele C++ ocolesc câteva din erorile frecvent întâlnite în cod ce abuzează de macro-instrucțiuni. Șabloanele au fost proiectate (gândite) ca să fie mai aplicabile decât macro-urile ***[1]****.*

---

3 *Ulrich Breymann, „**Designing components with the C++ STL**”, 2002.*

Există trei mari inconveniențe întâlnite la utilizarea șabloanelor:

1. *foarte multe compilatoare au avut un suport limitat pentru șabloane, astfel încât utilizarea acestora poate determina scăderea portabilității codului sursă;*
2. *aproape toate compilatoarele produc mesaje de eroare neproductive și derutante când sunt detectate erori în codul șablonului, aceasta poate forma ideea programării complicate a șabloanelor în C++;*
3. *fiecare utilizare a unui șablon poate determina generarea de către compilator a unei noi versiuni de cod pentru noua instanță a șablonului, deci utilizarea fără discernământ a șabloanelor poate duce la încărcarea codului, rezultând executabile excesiv de mari* ***[3]****.*

1.3 Componenta Containers în C++ STL

Un container este un obiect suport care stochează o colecție de alte obiecte (elementele sale). Acestea sunt implementate ca șabloane de clasă, ceea ce permite o mare flexibilitate în tipurile acceptate ca elemente. Containerul gestionează spațiul de stocare pentru elementele sale și oferă funcții membre pentru a le accesa, fie direct, fie prin iteratori (obiecte de referință cu proprietăți similare cu indicatorii).

Denumire	Descrierea
Containere secvențiale	*Acestea implementează structuri de date care pot fi accesate secvențial:* ***array*** *(matrice contiguă statică);* ***vector*** *(matrice contiguă dinamică);* ***deque*** *(coadă dublă);* ***forward_list*** *(listă legată/înlănțuită individual);* ***list*** *(listă legată/înlănțuită dublu).*
Containere asociative	*Acestea implementează structuri de date sortate care pot fi rapid căutate:* ***set****;* ***map****;* ***multiset****;* ***multimap****.*
Containere asociative neordonate	*Acestea implementează structuri de date nesortate (hash) care pot fi rapid căutate:* ***unordered_set****;* ***unordered_map****;* ***unordered_multiset****;* ***unordered_multimap****.*
Adaptoare pentru containere	*Oferă o interfață diferită pentru containerele secvențiale:* ***stack*** *(adaptează un container pentru a furniza stiva);* ***queue*** *(adaptează un container pentru a furniza coada);* ***priority_queue*** *(adaptează un container pentru a oferi o coadă cu priorități).*

1.4 Componenta Iterators în C++ STL

Un iterator este un obiect (asemenea unui pointer) care indică un element din interiorul containerului. Putem folosi iteratori pentru a ne deplasa prin conținutul containerului. Ei pot fi vizualizați ca ceva similar cu un indicator care sugerează o anumită locație și putem accesa conținutul din acea locație anume folosindu-i.

Iteratorii joacă un rol critic în conectarea algoritmului cu containerele, împreună cu manipularea datelor stocate în containere. Cea mai evidentă formă de iterator este un indicator. Un indicator poate indica elementele dintr-un masiv și poate itera prin ele folosind operatorul de incrementare și/sau decrementare (++/--). Nu toți iteratorii au o funcționalitate similară cu cea a indicatorilor.

În dependență de funcționalitate, iteratorii pot fi clasificați în trei categorii principale. Nu fiecare iterator este acceptat de toate containerele din STL, containere diferite acceptă iteratori diferiți, cum ar fi vectorii acceptă iteratori cu acces aleator, în timp ce listele acceptă iteratori bidirecționali.

Nr.	*Denumire container*	*Tipul de iterator*
1	*vector, deque*	*random-access*
2	*list, map, multimap, set, multiset*	*bidirecțional*
3	*stack, queue, priority_queue*	*niciun iterator acceptat*

Remarcă:

Tipurile de iteratori în baza funcționalității acestora mai pot fi clasificați și în cinci mari categorii: iteratori de intrare, iteratori de ieșire, iteratori de redirecționare, iteratori bidirecționali și iteratori cu access aleatoriu.

1.4.1 Avantajele iteratorilor

Există cu siguranță câteva modalități care arată că iteratorii ne sunt extrem de utili și ne încurajează să-i folosim la majoritatea aplicațiilor (produselor program). Unele dintre avantajele utilizării iteratorilor sunt:

1. ***Comoditate în programare:*** *este mai bine să utilizăm iteratori pentru a itera (naviga) prin conținutul containerelor și să parcurgem conținutul fără a trebui să ținem cont de nimic.*
2. ***Reutilizarea codului:*** *această reaplicare a secvenței de cod se va lua în considerare acum dacă facem dintr-un vector o listă, cu toate acestea, dacă am folosi iteratori pentru vectori pentru a accesa elementele, atunci doar schimbarea vectorului de listat în declarația iteratorului ar fi servit scopului, fără a face altceva. Deci, iteratorii acceptă reutilizarea codului, deoarece pot fi folosiți pentru a accesa elementele oricărui container.*
3. ***Procesare dinamică a containerului:*** *iteratorii ne oferă posibilitatea de a adăuga sau elimina dinamic elemente din container.*

1.4.2 Iteratori de intrare (Input)

Sunt cei mai slabi dintre toți iteratorii și au o funcționalitate limitată. Ei pot fi utilizați numai într-un algoritm cu o singură trecere, adică în acei algoritmi care procesează containerul secvențial, astfel încât niciun element să nu fie accesat de mai multe ori. După ce parcurgem definiția șablonului diferiților algoritmi STL, cum ar fi cei din namespace-ul STD: find, equal, count etc., trebuie să fi găsit definiția șablonului lor constând din obiecte de tip Input Iterator.

Deci, ce sunt și de ce sunt folosiți? Iteratorii de intrare sunt considerați a fi cei mai slabi, precum și cei mai simpli dintre toți iteratorii disponibili, pe baza funcționa-lității lor și a ceea ce se poate realiza folosindu-i. Aceștia sunt iteratorii care pot fi utilizață în operații de intrare secvențiale, în care fiecare valoare indicată de iterator este citită doar o dată și apoi iteratorul este incrementat.

Un lucru important care trebuie prezentat, este că iteratorii forward, bidirecționali și cu acces aleatoriu sunt, de asemenea, iteratori de intrare valabili.

**** Caracteristici importante*

1. ***Utilizare:*** *iteratorii de intrare pot fi folosiți numai cu algoritmi cu o singură trecere, adică algoritmi în care putem merge la toate locațiile din interval cel mult o dată. Spre exemplu, atunci când căutăm sau găsim orice element din interval, trecem prin locații cel mult o dată.*
2. ***Comparație egalitate / inegalitate:*** *un iterator de intrare poate fi comparat pentru egalitate cu un altul. Deoarece, iteratorii indică o anu-*

mită locație, astfel cei doi iteratori vor fi egali numai atunci când indică aceeași poziție, altfel nu.

3. ***Dereferențierea:*** *un iterator de intrare poate fi dereferențiat, folosind operatorul * și -> ca valoare, pentru a obține valoarea stocată în poziția indicată de iterator.*
4. ***Incrementabil:*** *un iterator de intrare poate fi incrementat, astfel încât să se refere la următorul element din secvență, folosind operatorul ++.*
5. ***Modificabil:*** *valoarea indicată de acești iteratori poate fi schimbată sau interschimbată.*

****** Limitări***

După studierea caracteristicilor esențiale, trebuie să cunoaștem și deficiențele (aspectele negative) care îl fac cel mai slab iterator dintre toate, care sunt menționate în următoarele puncte:

1. ***Accesând doar, fără atribuire:*** *una dintre cele mai mari deficiențe este că nu putem atribui nici o valoare locației indicate de acest iterator, poate fi utilizat doar pentru a accesa elemente și nu pentru a atribui elemente.*
2. ***Nu poate fi decrementat:*** *la fel cum putem folosi operatorul ++ cu iteratori de intrare pentru incrementarea lor, nu-i putem decrementa.*
3. ***Utilizare în algoritmi multi-pass:*** *deoarece este unidirecțional și poate merge doar înainte, prin urmare, astfel de iteratori nu pot fi utilizați în algoritmi multi-pass, în care trebuie să procesăm containerul de mai multe ori.*
4. ***Operatori relaționali:*** *deși, iteratorul de intrare poate fi utilizat cu operatorul de egalitate (==), dar nu poate fi utilizat cu alți operatori relaționali, cum ar fi <=.*
5. ***Operatori aritmetici:*** *similar operatorilor relaționali, aceștia nu pot fi folosiți nici cu operatori aritmetici precum +, - și așa mai departe. Aceasta înseamnă că operatorii de intrare se pot mișca doar într-o direcție (înainte și secvențial).*

1.4.3 Iteratori de ieşire (Output)

La fel ca iteratorii de intrare, ei sunt foarte limitați în funcționalitate și pot fi utilizați numai în algoritmul cu o singură trecere, dar nu pentru accesarea elementelor, ci pentru a fi atribuite elemente. După ce parcurgem

definiția șablonului diferiților algoritmi STL din namespace-ul STD, cum ar fi: copy, move, transform etc., trebuie să fi găsit definiția șablonului lor constând din obiecte de tip Output Iterator (iterator de ieșire).

Deci, ce sunt și unde sunt aplicați? Iteratorii de ieșire sunt considerați exact opusul iteratorilor de intrare, deoarece îndeplinesc funcția opusă a acestora. Lor li se pot atribui valori într-o succesiune, dar nu pot fi utilizați pentru a accesa valori, spre deosebire de iteratorii de intrare. Deci, putem spune că iteratorii de intrare și ieșire sunt complementari.

Un lucru important de reținut este faptul că iteratorii forward, bidirecționali și cu acces aleatoriu sunt iteratori de ieșire valabili.

**** Caracteristici importante*

1. ***Utilizare:** pot fi utilizați numai cu algoritmi cu o singură trecere, adică algoritmi în care putem merge la toate locațiile din interval cel mult o dată, astfel încât aceste locații pot fi dereferențiate sau pot fi atribuite doar o singură dată.*
2. ***Comparație, egalitate / inegalitate:** nu pot fi comparați pentru egalitate cu un alt iterator.*
3. ***Dereferențierea:** un iterator de intrare poate fi dereferențiat ca valoare, folosind operatorul * și ->, în timp ce un iterator de ieșire poate fi dereferențiat ca lvalue, pentru a furniza locația pentru stocarea valorii.*
4. ***Incrementabil:** poate fi incrementat, astfel încât să se refere la următorul element în ordine, folosind operatorul corespunzător.*
5. ***Modificabil:** valoarea indicată de acești iteratori poate fi schimbată sau interschimbată.*

**** Limitări*

După ce am studiat caracteristicile esențiale, trebuie să cunoaștem și deficiențele iteratorilor de ieșire, care sunt menționate în următoarele puncte:

1. ***Numai atribuire, fără acces:** una dintre cele mai mari deficiențe este că nu putem accesa iteratorii de ieșire ca valoare. Deci, un iterator de ieșire poate modifica elementul către care se îndreaptă, doar folosindu-se ca țintă pentru o atribuire.*
2. ***Nu poate fi decrementat:** la fel cum putem folosi operatorul de incrementare cu iteratori de ieșire, nu-i putem decrementa.*
3. ***Utilizare în algoritmi multi-pass:** deoarece, este unidirecțional și poate doar avansa, prin urmare, astfel de iteratori nu pot fi utilizați în*

algoritmi multi-pass, în care trebuie să ne deplasăm de mai multe ori prin container.

4. ***Operatori relaționali:*** *la fel ca iteratorii de ieșire nu pot fi utilizați cu operatori de egalitate, de asemenea, nu pot fi utilizați cu alți operatori relaționali precum "=".*
5. ***Operatori aritmetici:*** *similar operatorilor relaționali, aceștia nu pot fi folosiți nici cu operatori aritmetici precum "+", "-" ș.a.m.d. Aceasta înseamnă că operatorii de ieșire se pot deplasa doar într-o direcție înainte și secvențială.*

1.4.4 Iteratori de redirecționare (Forward)

Sunt mai mari în ierarhie decât iteratorii de intrare și ieșire, de asemenea includ toate caracteristicile prezente în acești doi iteratori. Dar, așa cum sugerează și numele lor, ei pot să se deplaseze doar într-o direcție. După ce ați parcurs definiția șablonului diferiților algoritmi STL, cum ar fi: search, search_n, lower_bound ș.a.m.d., trebuie să fi găsit definiția șablonului lor constând din obiecte de tip Forward Iterator.

Deci, ce sunt și unde sunt aplicați? Iteratorii forward sunt considerați a fi combinația de iteratori de intrare, precum și de ieșire. Ei oferă suport pentru funcționalitatea ambilor, prin permitere accesării și modificării valorilor.

Un lucru important de reținut este faptul că iteratorii bidirecționali și cu acces aleatoriu sunt iteratori forward valabili.

**** Caracteristici importante*

1. ***Utilizare:*** *efectuarea de operații pe un iterator direct, care nu poate fi referit nu face niciodată valoarea iteratorului său nereferențială, prin urmare, acest lucru permite algoritmilor care utilizează această categorie de iteratori să aplice (realizeze) mai multe copii ale unui iterator, pentru a trece de mai multe ori de aceleași valori. Deci, poate fi folosit în algoritmi multi-pass.*
2. ***Comparație, egalitate / inegalitate:*** *un iterator direct poate fi comparat pentru egalitate cu un alt iterator. Deoarece, iteratorii respectivi indică o anumită locație, doi iteratori vor fi egali numai atunci când indică aceeași poziție, altfel nicidecum.*
3. ***Dereferențierea:*** *deoarece un iterator de intrare poate fi dereferențiat, folosind operatorul "*" și "->" ca valoare și un iterator de ieșire poate fi*

dereferențiat de asemenea ca valoare, putem conclude că iteratorii de redirecționare pot fi utilizați în ambele scopuri.

4. ***Incrementabil:*** *un iterator direct poate fi incrementat, astfel încât să se refere la următorul element în ordine, folosind operatorul corespunzător. Este important să știm că putem aplica iteratori forward cu operatorul de incrementare, dar aceasta nu înseamnă că operatorul de decrementare poate fi de asemenea aplicat. Iteratorii* ***înainte*** *sunt unidirecționali și se pot deplasa doar în direcția înainte.*
5. ***Modificabil:*** *valoarea indicată de acești iteratori poate fi schimbată sau interschimbată.*

******* Limitări*

După studierea caracteristicilor esențiale, trebuie să cunoaștem și deficiențele, deși nu sunt atât de multe câte sunt în iteratorii de intrare sau de ieșire.

1. ***Nu poate fi decrementat:*** *nu-i putem decrementa, deși este mai mare în ierarhie față de iteratorii de intrare și ieșire, totuși nu poate depăși această deficiență.*
2. ***Operatori relaționali:*** *deși, iteratorii forward pot fi utilizați cu operatorul de egalitate"==", ei nu pot fi utilizați cu alți operatori relaționali precum "=".*
3. ***Operatori aritmetici:*** *similar operatorilor relaționali, aceștia nu pot fi folosiți nici cu operatori aritmetici precum "+", "-" ș.a.m.d. Aceasta înseamnă că operatorii de deplasare se pot deplasa doar într-o direcție înainte și secvențială.*
4. ***Utilizarea operatorului de dereferire offset ([ ]):*** *iteratorii forward nu acceptă operatorul de dereferire offset ([ ]), care este utilizat pentru acces aleatoriu.*

1.4.5 Iteratori bidirecționali (Bidirectional)

Au toate caracteristicile iteratorilor forward împreună cu faptul că depășesc dezavantajul iteratorilor forward, deoarece se pot deplasa în ambele direcții, de aceea numele lor este bidirecțional.

După ce parcurgeți definiția șablonului diferiților algoritmi STL, cum ar fi: reverse, next_permutation și reverse_copy ș.a.m.d.

Deci, ce sunt și unde sunt folosiți? Iteratorii bidirecționali sunt iteratori care pot fi utilizați pentru a accesa secvența elementelor într-un interval în ambele direcții (spre sfârșit și spre început). Aceștia sunt similari cu iteratorii înainte, cu excepția faptului că se pot deplasa și în direcția înapoi, spre deosebire de iteratorii înainte, care se pot deplasa doar în direcția înainte.

Trebuie remarcat faptul că containerele precum listă, hartă, multimap, set și multiset acceptă iteratori bidirecționali. Aceasta înseamnă că, dacă declarăm iteratori bidirecționali, la fel ca în cazul vectorilor și deque, ei sunt iteratori cu acces aleatoriu.

Un lucru important de reținut este că iteratorii cu acces aleatoriu sunt, de asemenea, iteratori bidirecționali valabili.

****** Caracteristici importante***

1. ***Utilizare:*** *deoarece, iteratorii forward pot fi utilizați în algoritmi multi-pass, prin urmare iteratorii bidirecționali pot fi folosiți și ei în algoritmi multi-pass.*
2. ***Comparație, egalitate / inegalitate:*** *un iterator bidirecțional poate fi comparat pentru egalitate cu un alt iterator. Deoarece, iteratorii indică o anumită locație, astfel doi iteratori vor fi egali numai atunci când indică aceeași poziție, altfel nicidecum.*
3. ***Dereferențierea:*** *deoarece un iterator de intrare poate fi dereferențiat, folosind operatorul "*" și "->" ca valoare și un iterator de ieșire poate fi dereferențiat ca valoare, astfel iteratorii de înaintare, care sunt combinația ambilor, pot fi aplicați în mod similar.*
4. ***Incrementabil:*** *un iterator bidirecțional poate fi incrementat, astfel încât să se refere la următorul element în ordine, folosind operatorul corespunzător.*
5. ***Decrementabil:*** *aceasta este caracteristica care diferențiază un iterator bidirecțional de un iterator direct. La fel cum putem folosi operatorul "++" cu iteratori bidirecționali, pentru incrementarea lor, le putem de asemenea diminua (decrementa).*
6. ***Modificabil:*** *valoarea indicată de acești iteratori poate fi schimbată sau interschimbată.*

****** Limitări***

După studierea caracteristicilor esențiale, trebuie să cunoaștem și deficiențele, deși nu sunt atât de multe câte sunt în iteratorii de intrare sau de ieșire.

1. ***Operatori relaționali:*** *deși, iteratorii bidirecționali pot fi utilizați cu operatori de egalitate "==", ei nu pot fi utilizați cu alți operatori relaționali precum "=".*
2. ***Operatori aritmetici:*** *similar operatorilor relaționali, aceștia nu pot fi folosiți nici cu operatori aritmetici precum "+", "-" ș.a.m.d. Aceasta înseamnă că iteratorii bidirecționali se pot deplasa în ambele direcții, dar secvențial.*
3. ***Utilizarea operatorului de dereferire offset ([ ]):*** *iteratorii bidirecționali nu acceptă operatorul de dereferire offset ([ ]), care este utilizat pentru acces aleatoriu.*

1.4.6 Iteratori cu acces aleatoriu (Random-Access)

Sunt cei mai puternici iteratori, aceștia nu se limitează la deplasarea secvențială, așa cum sugerează și numele lor, pot accesa aleatoriu orice element din container. Sunt cei ale căror funcționalități sunt identice cu indecșii. După ce parcurge definiția șablonului diferiților algoritmi STL precum nth_element, sort ș.a.m.d., găsim definiția șablonului lor constând din obiecte de tip Iterator cu acces aleator.

Deci, ce sunt și unde sunt folosiți? Iteratorii cu acces aleatoriu pot fi utilizați pentru a accesa elemente într-o poziție de decalare arbitrară față de elementul către poziția care se indică, oferind aceeași funcționalitate ca și pointerii. Aceștia sunt cei mai complecși din punct de vedere al funcționalității. Toate tipurile de pointer sunt iteratori valizi, cu acces aleatoriu.

Trebuie remarcat faptul că containerele precum vector, deque acceptă iteratori cu acces aleatoriu. Dacă declarăm iteratori standard pentru ei, atunci aceștia vor fi iteratori cu acces aleatoriu, la fel ca în cazurile: list, map, multimap, set și multiset.

****** Caracteristici importante***

1. ***Utilizare:*** *iteratorii cu acces aleatoriu pot fi folosiți în algoritmi multipass, adică algoritm care implică procesarea containerului de mai multe ori în diferite treceri.*

2. ***Comparație egalitate / inegalitate:*** *un iterator cu acces aleatoriu poate fi comparat pentru egalitate cu un alt iterator.*
3. ***Dereferențierea:*** *un iterator cu acces aleatoriu poate fi dereferențiat atât ca* ***rvalue****, cât și ca valoare de localizare (****lvalue****).*

Remarcă:

> *Termenii* ***lvalue*** *și* ***rvalue*** *nu sunt ceva care se întâlnește adesea în programarea C / C ++, de obicei nu este clar imediat ce înseamnă. Cel mai obișnuit loc în care se execută acești termeni se află în mesajele de eroare și de avertizare ale compilatorului.*
>
> *O valoare de localizare (****lvalue****) reprezintă un obiect care ocupă o anumită locație identificabilă în memorie (adică are o adresă).*
>
> *Prin urmare, din definiția de mai sus pentru lvalue,* ***rvalue*** *este o expresie care nu reprezintă un obiect care ocupă o anumită locație identificabilă în memorie.*

4. ***Incrementabil:*** *un iterator cu acces aleator poate fi incrementat, astfel încât să se refere la următorul element în ordine.*
5. ***Decrementabil:*** *la fel cum putem folosi operatorul „++” pentru iteratori cu acces aleatoriu pentru incrementarea lor, putem de asemenea și să-i decrementăm.*
6. ***Operatori relaționali:*** *deși, iteratorii bidirecționali nu pot fi utilizați cu operatori relaționali precum: „=”, dar iteratorii cu acces aleatoriu, care sunt mai mari în ierarhie, susțin toți acești operatori relaționali.*
7. ***Operatori aritmetici:*** *similar operatorilor relaționali, pot fi folosiți și cu operatori aritmetici precum „+”, „-” ș.a.m.d. Aceasta înseamnă că iteratorii cu acces aleator se pot deplasa atât într-o direcție, cât și întâmplător.*
8. ***Utilizarea operatorului de dereferire offset ([]):*** *iteratorii cu acces aleatoriu acceptă operatorul de dereferință specificat.*

1.4.7 Operaţiuni ale iteratorilor

Iteratorii sunt aplicați pentru a indica adresele de memorie ale containerelor STL. Aceștia reduc complexitatea și timpul de execuție al programului:

1. ***begin ():*** *- această funcție este utilizată pentru a reveni la poziția de început a containerului;*
2. ***end ():*** *- această funcție este utilizată pentru a reveni la poziția finală a containerului;*
3. ***advance ():*** *- această funcție este utilizată pentru a crește poziția iteratorului până la numărul specificat în argumentele sale;*
4. ***next ():*** *- returnează noul iterator pe care l-ar indica iteratorul după avansarea pozițiilor menționate în argumentele sale;*
5. ***prev ():*** *- returnează noul iterator pe care l-ar indica iteratorul după decrementarea pozițiilor menționate în argumentele sale;*
6. ***inserater ():*** *- această funcție este utilizată pentru a introduce elementele în orice poziție din container, acceptă 2 argumente (containerul și iteratorul) pentru poziționarea în care elementele trebuie inserate* ***[5]****.*

Secvența de cod C++ pentru prezentarea operațiilor: ***inserter()*** ***și*** ***advance ():***

Elaborăm o secvență de cod C++ cu implementarea STL, în care vom declara implicit un vector de elemente întregi. În antetul programului vom include componentele containerelor necesare ***[2]****. Pentru mai multe detalii e necesar să analizăm codul de mai jos împreună cu comentariile sale:*

```
#include<iostream>
#include<iterator> // pentru iterators
#include<vector> // pentru vectors
using namespace std;
int main(){
    vector<int> A = { 1, 2, 3, 4, 5 };
    vector<int> B = {10, 20, 30};
    // Declararea iteratorului la un vector
    vector<int>::iterator poz = A.begin();
    // Folosinm operația advance() pentru a seta poziția
    advance(poz, 3);
     // Copierea elementelor unui vector în alt vector folosind
operația inserter(), inserează B după poziția a 3-a în A.
    copy(B.begin(), B.end(), inserter(A,poz));
    // Afișăm rezultatele vectorului nou obținut
    cout << "Noul vector dupa inserarea elementelor este: ";
    for (int &x : A) cout << x << " ";
}
```

Rezultatul obținut în urma execuției secvenței de mai sus este:

```
Noul vector dupa inserarea elementelor este: 1 2 3 10 20 30 4 5
```

1.5 Componenta Algorithm în C++ STL

Spațiile de nume (namespace) ne permit să grupăm entități denumite care altfel ar avea sfera globală în sfere mai restrânse, oferindu-le sfera spațiului de nume. Aceasta permite organizarea elementelor programelor în diferite domenii logice menționate prin nume. Spațiul de nume (namespace) este o caracteristică adăugată în limbajul C++ și care nu este prezentă în C.

*Un spațiu de nume (namespace) este o regiune declarativă care oferă un domeniu de aplicare identificatorilor (nume de tipuri, funcții, variabile etc.) din interiorul acestuia. Sunt permise mai multe blocuri de spațiu de nume (namespace) cu același nume. Toate declarațiile din aceste blocuri sunt declarate în domeniul de aplicare numit **[3]**, **[5]**.*

*Operațiile prezentate mai jos conform categoriilor sale, aparțin spațiului de nume **std**, adică pentru a le utiliza într-un program C++, este necesar să includem acest spațiu de nume: **using namespace std.***

Nr.	Operația	Descrierea
Operații de secvență imuabile		
1	all_of	*starea testului pe toate elementele din interval;*
2	any_of	*testează dacă vre-un element din gamă îndeplinește condiția;*
3	none_of	*testează dacă niciun element nu îndeplinește condiția;*
4	for_each	*aplicăm funcția într-un interval;*
5	find	*găsim valoarea într-un interval;*
6	find_if	*găsim elementul într-un interval;*
7	find_if_not	*găsim elementul într-un interval (condiție negativă);*
8	find_end	*găsim ultima subsecvență în interval;*
9	find_first_of	*găsim elementul din set în interval;*

Nr.	Operaţia	Descrierea
10	adjacent_find	*găsim elemente adiacente egale în interval;*
11	count	*numărăm aparițiile valorii în interval;*
12	count_if	*returnăm numărul de elemente în condiții satisfăcătoare;*
13	mismatch	*returnăm prima poziție în care două intervale diferă;*
14	equal	*testăm dacă elementele din două game sunt egale;*
15	is_permutation	*testăm dacă intervalul este permutarea altuia;*
16	search	*interval de căutare pentru subsecvență;*
17	search_n	*interval de căutare pentru element;*
	Operaţii de secvenţă muabile	
1	copy	*copiem gama de elemente;*
2	copy_n	*elemente de copiere;*
3	copy_if	*copiem anumite elemente ale intervalului;*
4	copy_backward	*copiem gama de elemente înapoi;*
5	move	*mutăm gama de elemente;*
6	move_backward	*mutăm gama de elemente înapoi;*
7	swap	*schimbăm / interschimbăm valorile a două obiecte;*
8	swap_ranges	*valori de schimb pentru două intervale;*
9	iter_swap	*schimbăm valorile obiectelor îndreptate către doi iteratori;*
10	transform	*gama de transformare;*
11	replace	*înlocuim valoarea în interval;*
12	replace_if	*înlocuimvalorile din interval;*
13	replace_copy	*copiem valoarea înlocuind intervalul;*
14	replace_copy_if	*copiem valoarea de înlocuire a intervalului;*

Nr.	Operaţia	Descrierea
15	fill	*umplem intervalul cu valoare;*
16	fill_n	*completăm secvența cu valoare;*
17	generate	*generăm valori pentru interval cu funcție;*
18	generate_n	*generăm valori pentru secvență cu funcție;*
19	remove	*eliminăm valoarea din interval;*
20	remove_if	*eliminăm elementele din interval;*
21	remove_copy	*copiem valoarea eliminând intervalul;*
22	remove_copy_if	*copiem intervalul eliminând valorile;*
23	unique	*eliminăm duplicatele consecutive din interval;*
24	unique_copy	*copiem gama eliminând duplicatele;*
25	invers	*interval invers;*
26	reverse_copy	*copiem intervalul inversat;*
27	rotate	*rotim la stânga elementele în raza de acțiune;*
28	rotate_copy	*copiem zona rotită la stânga;*
29	random_shuffle	*rearanjăm aleatoriu elementele din interval;*
30	shuffle	*rearanjăm aleatoriu elementele din interval folosind generatorul;*
Operaţii de partiţie		
1	is_partitioned	*testăm dacă intervalul este partiționat;*
2	partition	*gama de partiții în două;*
3	stable_partition	*gama de partiții în două - ordonare stabilă;*
4	partition_copy	*interval de partiție în două;*
5	partition_point	*obținem punctul de partiție;*
Operaţii de triere (sortare, clasificare, selecţionare)		
1	sort	*sortăm elementele în interval;*

Nr.	Operaţia	Descrierea
2	stable_sort	*sortăm elementele păstrând ordinea echivalenților;*
3	partial_sort	*sortăm parțial elementele din interval;*
4	partial_sort_copy	*copiem și sortăm parțial intervalul;*
5	is_sorted	*verificăm dacă intervalul este sortat;*
6	is_sorted_until	*găsim primul element nesortat din interval;*
7	nth_element	*sortăm elementul al n-lea în interval;*
Căutare binară (funcţionează pe intervale partiţionate / sortate)		
1	lower_bound	*întoarce iteratorul la limita inferioară;*
2	upper_bound	*întoarce iteratorul la limita superioară;*
3	equal_range	*obținem subrange de elemente egale;*
4	binary_search	*testăm dacă valoarea există în ordine sortată;*
Combină (funcţionează pe intervale sortate)		
1	merge	*combinăm intervale sortate;*
2	inplace_merge	*combinăm intervale sortate consecutive;*
3	include	*testăm dacă intervalul sortat include un alt interval sortat;*
4	set_union	*unim două intervale sortate;*
5	set_intersection	*intersectăm a două intervale sortate;*
6	set_difference	*diferența a două intervale sortate;*
7	set_symmetric_difference	*diferența simetrică a două intervale sortate;*
Operaţii Heap		
1	push_heap	*împingem elementul în intervalul heap;*
2	pop_heap	*extragem elementul din intervalul heap;*
3	make_heap	*facem heap din interval;*

Nr.	Operaţia	Descrierea
4	sort_heap	*sortăm elementele heap-ului;*
5	is_heap	*testăm dacă intervalul este heap;*
6	is_heap_until	*găsim primul element care nu este în ordine heap;*
7	max	*returnăm cel mai mare element;*
8	minmax	*returnăm cele mai mic și mai mare element;*
9	min_element	*returnmă cel mai mic element din interval;*
10	max_element	*returnăm cel mai mare element din interval;*
11	minmax_element	*returnăm cele mai mic și cel mai marieelement din interval;*
Alte operaţiuni		
1	lexicographical_compare	*comparație mai puțin decât lexicografică;*
2	next_permutation	*transformăm intervalul în următoarea permutare;*
3	prev_permutation	*transformăm intervalul în permutarea anterioară.*

1.6 Prezentare generală pentru C++20

Grupul de utilizatori C++ este destul de mare. De la apariția C++ 98 până la finalizarea oficială a C++ 11, a acumulat peste un deceniu.

C++ 14/17 este un important complement și optimizare pentru C++ 11, iar C++ 20 aduce acest limbaj la ușa modernizării. Caracteristicile extinse ale tuturor acestor noi standarde sunt date limbajului C++.

Infuzat cu o nouă vitalitate. Programatorii C++, care încă folosesc C++ tradițional (C++ 98), pot fi chiar uimiți de faptul că nu folosesc același limbaj în timp ce citesc codul modern C++ [4].

4 *Proiectul „**The Standard**” , https://isocpp.org . Accesat, 17 septembrie 2020*

C++20 este numele revizuirii standardului ISO / IEC pentru limbajul de programare C ++ după C ++ 17[5]. Standardul a fost finalizat tehnic [6] de către WG21 la reuniunea de la Praga din februarie 2020.

C++20 se află în prezent în lucrări editoriale finale, după ce un proiect a fost aprobat la 4 septembrie 2020. C++20 adaugă mai multe funcții majore noi decât C ++ 14 sau C ++ 17. Planificat inițial pentru lansare în februarie anul trecut, C ++ 20 a primit acum aprobarea tehnică finală și va fi publicat în mod previzibil până la sfârșitul anului.

C++20 va include module, coroutine și concepte printre noile sale caracteristici majore. Modulele introduc un nou mod de încapsulare a codului, care are ca scop, de asemenea, să facă inutile fișierele antet.

*Semnificația modulelor nu poate fi minimizată, a scris un expert proeminent al limbajului C++, Herb Sutter: **„Aceasta este prima dată în aproximativ 35 de ani când C++ a adăugat o nouă caracteristică în care utilizatorii pot defini o limită de încapsulare numită."**[7]*

Împreună cu variabilele, funcțiile și clasele, modulele oferă programatorilor o modalitate de a denumi o entitate care ascunde un fel de implementare. Până în prezent, variabilele ar putea fi utilizate pentru a încapsula valori și funcții pentru a încapsula comportamentul, în timp ce clasele încapsulează starea și comportamentul. Modulele oferă un mecanism pentru a încapsula variabile, funcții și clase împreună.

*Herb Sutter a spus: **„Acesta este un motiv fundamental care stă la baza modulelor care permit îmbunătățiri suplimentare care pot urma acum în viitoarea evoluție C ++."***

*Acesta este modul în care puteți defini un modul simplu care exportă o funcție și utilizarea acesteia într-un fișier diferit **[7]**:*

```
// math.cppm
export module mod1;
export int identity(int arg) {
    return arg;
}
```

5 *Sutter Herb - „**P1000R3: programare C ++ IS**" (PDF). Accesat, 23-09-2020*

6 *Dusíková Hana - „**N4817: Invitație și informații pentru întâlnirea din Praga 2020**". Accesat, 27.09.2020.*

7 *Resursă web: https://www.infoq.com/news/2020/09/cpp-20-final/, Accesat, 2 octombrie 2020*

```
// main.cpp import mod1;
int main() {
    identity(100);
}
```

Conceptele sunt o extensie a șabloanelor care permit definirea predicatelor în timp de compilare pentru parametrii șablonului. Conceptele sunt menite să modeleze categoriile semantice mai degrabă decât restricțiile sintactice.

*Biblioteca standard include deja o serie de concepte predefinite, cum ar fi Integral, Assignable, StrictTotallyOrdered, MoveConstructible, Callable și multe altele **[9]**. De exemplu, conceptul Integral este definit astfel:*

```
template <class T>
concept bool Integral() {
      return std::is_integral<T>::value;
}
```

Odată ce ați definit un concept, îl puteți folosi într-o definiție de șablon ca aceasta:

```
template<Integral T> // stenografie pentru: template<typename T>
necesită Integral<T>()
T foo(T a, T b) {
 ...
}
Integral foo(Integral&); // chiar mai scurt
```

C++20 include, de asemenea, o serie de modificări mai mici, inclusiv extensia de legare structurată și captura de referință, inițializarea agregatelor folosind (), polymorphic_allocator și multe altele.

În timp ce C++20 își face ultima cale de a deveni actualul standard C++, comitetul ISO a început deja să lucreze la C++23, care ar putea include o bibliotecă standard modulară, executanți, reflecție, potrivire a modelelor și contracte, deși pandemiile actuale și restricțiile la întâlnirea față în față vor reduce probabil cantitatea de muncă care poate fi făcută.

Grupul de utilizatori C++ este destul de mare. De la apariția C++ 98 până la finalizarea oficială a C++ 11, a acumulat peste un deceniu. C++ 14/17 este

un important complement și optimizare pentru C ++ 11, iar C++ 20 aduce acest limbaj la ușa modernizării.

Caracteristicile extinse ale tuturor acestor noi standarde sunt date limbajului C++. Infuzat cu o nouă vitalitate. Programatorii C++, care încă folosesc C++ tradițional (această carte se referă la C++ 98 și standardele sale anterioare ca C++ tradițional), pot fi chiar uimiți de faptul că nu folosesc același limbaj în timp ce citesc codul C++ modern.

Exemplu de problemă rezolvată cu C++ 20 STL

Fie avem un masiv unidimensional de numere întregi. Să se determine cel mai mare al N-lea element din masiv (listă de numere). Exemplu: A={ 7, 4, 6, 3, 9, 1} și N=2, atunci soluția problemei ar fi: Al 2-lea cel mai mare element este 7.

Rezolvare:

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;
int Nmaxim(vector<int> const &A, int N){
	priority_queue<int, vector<int>, greater<>>
	pq(A.begin(), A.begin() + N);
	for (int i=N; i<A.size();i++){
		if (A[i]>pq.top()){
			pq.pop(); pq.push(A[i]);
		}
	}
	return pq.top();
}
int main(){
	vector<int> A = { 7, 4, 6, 3, 9, 1 };
	int N = 2;
	cout<<"Al "<<N<<"-lea element maxim: "<<Nmaxim(A, N);
}
```

2. LISTE SIMPLU ÎNLĂNȚUITE

2.1 Introducere

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 afară de informația utilă, adresa următorului element. Această organizare permite numai acces secvențial la elementele listei.

Pentru accesarea listei trebuie cunoscută adresa primului element (numită capul listei). Elementele următoare sunt accesate parcurgând lista.

O listă liniară simplu înlănțuită conține elemente (noduri) ale căror valori constau din două părți: informația utilă și informația de legătură. Informația utilă reprezintă informația propriu-zisă memorată în elementul liste (numere, șiruri de caractere, etc.), iar informația de legătură precizează adresa următorului element al listei.

Lista simplu înlănțuită poate fi reprezentată grafic astfel:

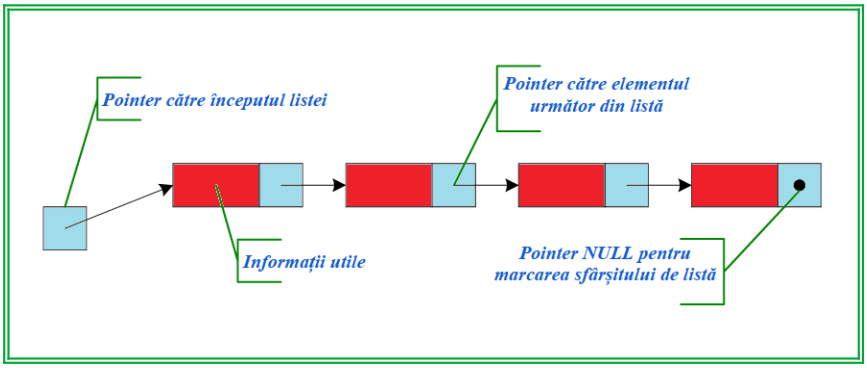


Figura 1.1 – Reprezentarea grafică a unei liste liniare simplu înlănțuite

2.2 Structura unei liste simplu înlănțuite

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

```
// Datele asociate unui element (nod) dintr-o listă
typedef int Date;
```

În cazul în care se dorește memorarea unui alt tip de date, trebuie schimbată doar declarația aliasului Date. Pentru memorarea listei se foloseste o structura autoreferita. Acesta structura va avea forma:

```
struct Element{
     Date valoare; // datele efective memorate
     Element* next; // legatura către nodul următor
};
```

O metodă mai des întâlnită de prezentatare a unei liste liniare simplu înlănțuită alocată dinamic:

```
struct Element{
     int valoare;
     Element* next;
};
```

*Câmpul **valoare** al tipului **Element** reprezintă informația utilă – în acest caz un număr întreg, iar câmpul **next** este de tip pointer la **Element** și reprezintă informația de legătură.*

*În program vom folosi o variabilă de tip pointer (de exemplu **first**) pentru a memora adresa primului element al listei și fiecare element al listei, începând cu primul, va memora în câmpul **next** adresa elementului următor. Excepție face ultimul element al listei care va memora în câmpul **next** valoarea **NULL**.*

*La început **first** va avea valoarea **NULL**, cu semnificația că lista este vidă. Dacă la un moment dat lista redevine vidă (de exemplu se șterg toate elementele ei) variabila **first** va avea valoarea **NULL**.*

Elementele listei sunt variabile dinamice, create cu ajutorul operatorului C++ ***new*** *și gestionate prin intermediul pointerilor. Variabila* ***first*** *este de tip pointer, dar este (în cele ce urmează) statică. Fiind variabile dinamice, pentru elementele listei se alocă memorie în HEAP.*

Informațiile de legătură ocupă memorie. Spațiul de memorie ocupat de un pointer depinde de versiunea compilatorului folosit; în general este de 4 octeți. Astfel, fiecare element al unei liste de tipul de mai sus va ocupa în memorie 4+4=8 octeți. Accesul la un nod al listei se face prin parcurgerea nodurilor care îl preced.

O secvență C++ care conține declarațiile corespunzătoare poate fi:

```
struct Element{
     int valoare;
     Element* next;
};
Element* first = NULL;
```

În continuare vom prezenta principalele operații:

- *crearea unui element nou;*
- *adăugarea unui element la începutul (sfârșitul) listei;*
- *parcurgerea listei;*
- *ștergere unui element din listă etc.*

2.3 Operaţii principale efectuate

2.3.1 Crearea unui element nou

Numeroase operații cu liste solicită crearea unui nou element (nod). Pentru aceasta trebuie să ținem cont de următoarele:

1. *Nodurile sunt variabile dinamice. Crearea unui nou nod înseamnă crearea unei variabile dinamice. Acest lucru se face cu ajutorul operatorului C++* ***new****, care are ca rezultat adresa variabilei nou create. Aceasta va fi memorată într-un pointer de tip* ***Element*****. Să-l numim* ***p: Element* p = new Element;***
2. *Nodurile sunt variabile de tip structură, cu câmpurile* ***info*** *și* ***next****. Accesul la câmpuri se va face prin intermediul pointerilor, cu ajutorul operatorului ->, astfel:* ***p->valoare*** *și* ***p->next****. Accesul la câmpuri se poate face și după dereferențierea pointer-ului:* ***(* p). valoare*** *și* ***(* p).next****.*
3. *Nodul nou creat va fi inclus într-o listă.* ***p->next*** *va memora adresa următorului element, sau* ***NULL*** *dacă nu există următorul element!*

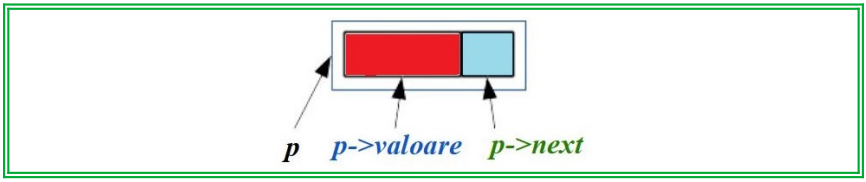


Figura 1.2 – Reprezentarea grafică a unui nod creat

Rezumat:

- ***p*** *este pointer la* ***Element****, care este de tip* ***Element****;*
- ****p*** *este variabila de tip* ***Element*** *– este nod din listă;*
- ***p-> valoare*** *este informația utilă din nodul listei, de tip* ***int****;*
- ***p->next*** *este pointer. Memorează adresa elementului următor!*

```
nod * p = new nod;
p->info = ..... ; // cin >> p->info;
p->urm = NULL;
```

2.3.2 Parcurgerea și afișarea listei

Lista este parcursă pornind de la pointer spre primul element și avansând folosind pointerii din structură până la sfârșitul listei (pointer NULL).

```
// Parcurgere și afișare pentru o listă simplă
void Afisare(Element* cap) {
     // Cât timp mai avem elemente în listă
     while (cap != NULL) {
          // Afișează elementul curent
          cout << cap->valoare << endl;
          // Avansează la elementul următor
          cap = cap -> urmator;
     }
}
```

2.3.3 Inserarea (adăugarea) unui element în listă

Pentru inserarea unui element în listă, există câteva modalități:

- *inserarea la începutul listei;*
- *inserarea la sfârșitul listei;*
- *inserarea după un element dat.*

Inserarea la începutul listei este cazul cel mai simplu: trebuie doar alocat elementul, legat de primul element din listă și repoziționarea capului listei.

```
void InserareInceput(Element* &cap, Date val) {
    // Alocare nod și inițializare valoare
    Element *elem = new Element; elem->valoare = val;
    // Legare nod în listă
    elem->urmator = cap;
    // Mutarea capului listei
    cap = elem;
}
```

Inserarea la sfârșitul listei este cazul în care trebuie întâi parcursă lista și după aceea adăugat elementul și legat de restul listei.

```
void InserareSfarsit(Element* &cap, Date val) {
     // Alocare și inițializare nod
     Element *elem = new Element; elem->valoare = val;
     elem->urmator = NULL;
     // Dacă avem lista vidă
     if (cap == NULL)
     // Doar modificăm capul listei
     cap = elem;
     else {
          // Parcurgem lista până la ultimul nod
          Element *nod = cap;
          while (nod->urmator != NULL)
          nod = nod -> urmator;
          // Adăugăm elementul nou în listă
          nod->urmator = ele m;
     }
}
```

Inserarea unui elemen după un element dat (stabilit) va fi reprezentată de următoarea secvență de cod:

```
void InsInt(Element* &cap, Element* p, Date val) {
     // Alocare și inițializare nod
     Element *elem = new Element; elem->valoare = val;
     elem->urmator = NULL;
     // Lista vidă
     if (cap == NULL) {
        cap = ele m; return;
     }
     // Inserare la începutul listei
     if (cap == p) {
        elem->urmator = cap ; cap = ele m; return;
     }
     // Inserare în interiorul listei
     elem->urmator = p->urmator; p->urmator = elem;
}
```

2.3.4 Căutarea unui element în listă

Căutarea unui element dintr-o listă presupune parcurgerea listei pentru identificarea nodului în funcție de un criteriu. Cele mai uzuale criterii sunt cele legate de poziția în cadrul listei și de informațiile utile conținute de nod.

Rezultatul operației este adresa primului element găsit sau NULL. Pentru căutarea unui element în listă, există câteva modalități:

- *căutarea după poziție;*
- *căutarea după valoare.*

Pentru căutarea unui element după poziție, se avansează pointerul cu numărul de poziții specificat:

```
Element* CautarePozitie(Element* cap, int pozitie) {
     int i = 0; // Poziția curentă
     // Parcurge lista până la poziția cerută sau
     // până la sfârșitul listei
     while (cap != NULL && i < pozitie) {
           cap = cap -> urmator; i++;
     }
     // Dacă lista conține elementul
     if (i == poziție) return cap;
     else return NULL;
}
```

Pentru căutarea unui element după valoare, se parcurge lista până la epuizarea acesteia sau identificarea elementului:

```
Element* CautareValoare(Element* cap, Date val) {
     // Parcurge lista până la găsirea elementului sau
     // epuizarea listei
     while (cap != NULL && cap->valoare != val)
     cap = cap -> urmator;
     return cap;
}
```

2.3.5 Ștergerea unui element din listă

Pentru ștergerea unui element din listă, există câteva modalități:

- *ștergerea unui element din interiorul listei;*
- *ștergerea unui element de pe o anumită poziție;*
- *ștergerea după o valoare.*

Pentru ștergerea unui element din interiorul listei (diferit de capul listei), avem nevoie de adresa predecesorului elementului de șters. Se modifică legăturile în sensul scurtcircuitării elementului de șters, după care se elibe-rează memoria corespunzătoare elementului de șters.

```
void StergereElementInterior(Element* predecesor) {
   // Salvăm referința la elementul de șters
   Element* deSters = predecesor->urmator;
   // Scurcircuităm elementul și îl ștergem
   predecesor->urmator = predecesor->urmator->urmator;
   delete deSters;
}
```

Pentru ștergerea unui element de pe o anumită poziție trebuie să ținem cont de faptul că dacă elementul este primul din listă, atunci se modifică capul listei, altfel se caută elemental și se șterge folosind funcția definită anterior.

```
void StergerePozitie(Element* &cap, int pozitie) {
if (cap == NULL) return;
// Dacă este primul, atunci îl ștergem și mutăm capul
if (pozitie == 0) {
    Element* deSters = cap; cap = cap -> urmator;
    delete deSters; return;
}
// Dacă este în interor, atunci folosim funcția de ștergere
Element* predecesor = CautarePozitie(cap, pozitie-1); Sterg-
ereElementInterior(predecesor);
}
```

Pentru ștergerea unui element după o valoare, trebuie să ținem cont de faptul că se caută predecesorul elementului și se folosește funcția de ștergere ele-ment.

```
void StergereValoare(Element* &cap, Date val) {
// Dacă lista e vidă, nu facem nimic
if (cap == NULL) return;
// Dacă este primul, atunci îl ștergem și mutăm capul
if (cap->valoare == val) {
   Element* deSters = cap; cap = cap -> urmator;
   delete deSters; return;
}
// Căutăm predecesorul
Element* elem = cap;
while (elem->urmator != NULL && elem->urmator->valoare != val)
elem = ele m- >urmator;
// Dacă a fost găsit, atunci îl ștergem
if (elem->urmator != NULL) StergereElementInterior(elem);
}
```

Remarcă:

- *Dacă vă mai aduceți aminte cum se realizează un exemplu de fișier antet, creați un asemenea fișier pentru structura dinamică respectivă, incluzând operațiile studiate în acest compartiment.*
- *Pentru crearea unui asemenea fișier, este necesar să analizați pașii prezentați în exemplele din anexa 1.*
- *De asemenea la dorință se poate de îmbunătățit acest fișier antet cu alte subprograme pe care le veți aplica la rezolvarea problemelor.*

2.4 Modele de probleme rezolvate

Problema 1 – Registru de studenți

Să se scrie un program pentru evidenţa studenţilor unui an de studiu. Pentru rezolvarea problemei se vor folosi liste simplu înlănțuite. Informațiile de interes sunt numele studentului şi grupa. Programul va implementa următoarele operaţii:

1. *Introducerea unui student;*
2. *Afisarea ordonată a studenţilor;*
3. *Căutare unui student;*
4. *Ştergerea unui student.*

Prezentarea soluției problemei în limbajul C++

```
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
using namespace std;
typedef struct student {
    char *nume;
    float grupa;
    struct student *urm;
}nod;
///****************** OPERAȚIA 1 ******************
nod *adauga(nod*prim,char *nume,float grupa) {
    nod *q,*p;p=(nod*)malloc(sizeof(nod));
    p->nume=(char*)malloc(strlen(nume)+1);
    strcpy(p->nume,nume); p->grupa=grupa; p->urm=NULL;
    if(p==NULL || p->nume==NULL) {
        cout<<"Eroare la alocare!"; exit(0);
    }
    if(prim==NULL) return p;
    else if(strcmp(prim->nume,p->nume)>0){
        p->urm=prim; return p;
    }
    else {
```

```
       q=prim;
       while(q->urm!=NULL && strcmp(q->urm->nume,p->nume)<0)
       q=q->urm; p->urm=q->urm; q->urm=p; return prim;
    }
}
///****************** OPERAȚIA 2 ******************
void afisare(nod *prim){
    nod *q; q=prim; while(q!=NULL){
        cout<<"\n"<<q->nume<<"\t\t\t"<<q->grupa; q=q->urm;
    }
}
///****************** OPERAȚIA 3 ******************
nod* cautare(nod *prim,char *nume){
    nod *q; q=prim;
    while(q!=NULL && strcmp(q->nume,nume)!=0) q=q->urm;
    return q;
}
///****************** OPERAȚIA 4 ******************
nod* stergere(nod *prim,char *nume){ nod *q,*p;
    if(prim!=NULL) {
        if(strcmp(prim->nume,nume)==0) {
            q=prim; prim=prim->urm; free(q->nume);
            free(q); return prim;
        }
        q=prim;
        while(q->urm!=NULL && strcmp(q->urm->nume,nume)!=0)
        q=q->urm;
        if(q->urm!=NULL && strcmp(q->urm->nume,nume)==0) {
            p=q->urm; q->urm=q->urm->urm; free(p->nume);
            free(p);
        }
        return prim;
    }
    else return prim;
}
///****************** PROGRAMUL PRINCIPAL **********
int main(){
    int opt; nod *prim,*p;
    char nume[10];
    float grupa; prim=NULL;
    while(1){
      system("CLS");
```

```
      cout<<"\n 1. Introducerea unui student";
      cout<<"\n 2. Afisarea alfabetica a studentilor";
      cout<<"\n 3. Cautarea unui student";
      cout<<"\n 4. Stergerea unui student";
      cout<<"\n 5. Iesire\n"; cin>>opt; fflush(stdin);
      switch(opt) {
      case 1:cout<<"Numele:"; gets(nume);
      cout<<"Grupa:"; cin>>grupa;
      prim=adauga(prim,nume,grupa); break;
      case 2:afisare(prim); break;
      case 3:cout<<"Introduceti numele studentului cautat:";
      gets(nume); p=cautare(prim,nume);
      if(p==NULL)
      cout<<"Studentul nu se gaseste in lista!";
      else cout<<p->nume<<" "<<p->grupa; break;
      case 4:cout<<"Introduceti numele studentului:";
      gets(nume); prim=stergere(prim,nume);
      afisare(prim); break;
      case 5:exit(1); default:cout<<"\n1..5!";
    }
    getchar();
    }
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
 1. Introducerea unui student
 2. Afisarea alfabetica a studentilor introdusi
 3. Cautarea unui student
 4. Stergerea unui student
 5. Iesire
```

Spre exemplu avem 4 studenți introduși: Ionela din grupa 101, Gabriela din grupa 103, Romică din grupa 102 și Alexa din grupa 105. Atunci când vom selecta opțiunea 2 din meniu, vom avea:

```
Alexa                   105
Gabriela                103
Ionela                  101
Romica                  102
```

Problema 2 – Numărul de apariții ale cheii

Fie avem o listă simplu înlănțuită și o cheie (un număr), numărați numărul de apariții ale cheii date în listă. Se va utiliza un algoritm recursiv.

De exemplu, dacă elementele din listă sunt: 1-> 2-> 1-> 2-> 1-> 3-> 1-> 4-> 5-> 2-> 9 și cheia dată este 1, atunci ieșirea ar trebui să fie 4.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
struct Nod {
    int data;
    struct Nod* next;
};
// variabila globala pentru numararea frecventei elementului
dat k (cheia)
int frecventa = 0;
/* Având o referință la capul unei liste și o variabilă de tip
int, împingem un nou nod în fața listei. */
void push(struct Nod** head_ref, int new_data){
    struct Nod* new_nod = (struct Nod*)malloc(sizeof(struct
Nod));
    new_nod->data = new_data;
    new_nod->next = (*head_ref);
    (*head_ref) = new_nod;
}
/* Numărăm numărul de apariții ale unui nod într-o listă */
int count(struct Nod* head, int key){
    if (head == NULL) return frecventa;
    if (head->data == key) frecventa++;
    return count(head->next, key);
}
/* Programul principal*/
int main(){
    int cheia;
    /* Începem cu lista goală */
    struct Nod* head = NULL;
```

```
    /* Utilizăm push() pentru a construi lista:
    1->2->1->2->1->3->1->4->5->2->9 */
    push(&head, 1); push(&head, 2); push(&head, 1);
    push(&head, 2); push(&head, 1); push(&head, 3);
    push(&head, 1); push(&head, 4); push(&head, 5);
    push(&head, 2); push(&head, 9);
    cout << "Introdu numarul cheie ce trebui de verificat: ";
    cin >> cheia;
    /* Verificăm funcția de numărare */
    cout << "Numarul de aparitii ale cheii " << cheia << "
este: ";
    cout << count(head,cheia) << endl;
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Introdu numarul cheie ce trebuie de verificat: 1
Numarul de aparitii ale cheii 1 este: 4
```

```
Introdu numarul cheie ce trebuie de verificat: 2
Numarul de aparitii ale cheii 2 este: 3
```

```
Introdu numarul cheie ce trebuie de verificat: 3
Numarul de aparitii ale cheii 3 este: 1
```

```
Introdu numarul cheie ce trebuie de verificat: 4
Numarul de aparitii ale cheii 4 este: 1
```

```
Introdu numarul cheie ce trebuie de verificat: 5
Numarul de aparitii ale cheii 5 este: 1
```

Problema 3 – Lista palindrom

Fie avem o listă simplu înlănțuită de caractere, scrieți un subprogram care returnează adevărat dacă lista dată este un palindrom, altfel fals. Se va utiliza un algoritm recursiv.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
struct nod {
    char data;
    struct nod* next;
};
// Parametrii inițiali ai acestei funcții sunt &head și head
bool isPalindromeUtil(struct nod** left, struct nod* right){
    /* Oprim recursivitatea când dreapta devine NULL */
    if (right == NULL) return true;
    /* Dacă este palindrom, atunci nu este nevoie să verificăm
elementul curentul în stânga și în dreapta, returnăm fals */
    bool isp = isPalindromeUtil(left, right->next);
    if (isp == false) return false;
    /* Verificăm valorile curente la stânga și la dreapta */
    bool isp1 = (right->data == (*left)->data);
    /* Mutăm la stânga la nodul următor */
    *left = (*left)->next;
    return isp1;
}
bool isPalindrome(struct nod* head){
    isPalindromeUtil(&head, head);
}
/* Împingem un nod în lista. Reținem că această funcție schimbă
capul listei*/
void push(struct nod** head_ref, char new_data){
    /* Alocăm nodul */
    struct nod* new_nod = (struct nod*)malloc(sizeof(struct
nod));
    /* Introducem datele */
    new_nod->data = new_data;
    /* Anexăm vechea listă a noului nod */
```

```
    new_nod->next = (*head_ref);
    (*head_ref) = new_nod;
}
// O funcție utilitară pentru a imprima o anumită listă
void printList(struct nod* ptr){
    while (ptr != NULL) {
        cout << ptr->data << "->";
        ptr = ptr->next;
    }
    cout << "NULL \t" ;
}
/* Programul principal pentru a testa funcțiile de mai sus*/
int main(){
    /* Începem cu lista goală */
    struct nod* head = NULL;
    char str[] = "abacicaba";
    int i;
    for (i = 0; str[i] != '\0'; i++) {
        push(&head, str[i]); printList(head);
        isPalindrome(head) ? cout << " + Este palindrom!\n" :
cout << " - Nu este palindrom!\n";
    }
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
a->NULL          + Este palindrom!
b->a->NULL       - Nu este palindrom!
a->b->a->NULL    + Este palindrom!
c->a->b->a->NULL         - Nu este palindrom!
i->c->a->b->a->NULL      - Nu este palindrom!
c->i->c->a->b->a->NULL   - Nu este palindrom!
a->c->i->c->a->b->a->NULL        - Nu este palindrom!
b->a->c->i->c->a->b->a->NULL     - Nu este palindrom!
a->b->a->c->i->c->a->b->a->NULL          + Este palindrom!
```

Problema 4 – Intersecția a două liste simple

Fie avem două liste sortate în ordine crescătoare, creați și returnați o nouă listă reprezentând intersecția celor două liste. Noua listă trebuie făcută cu propria memorie - listele originale nu trebuie schimbate. Se va utiliza un algoritm recursiv.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
struct Nod {
    int data;
    struct Nod* next;
};
struct Nod* sortamIntersectia(struct Nod* a, struct Nod* b){
    /* Cazul de bază */
    if (a == NULL || b == NULL) return NULL;
    /* Dacă ambele liste nu sunt goale, avansăm lista mai mică
și apelăm recursiv */
    if (a->data<b->data) return sortamIntersectia(a->next, b);
    if (a->data>b->data) return sortamIntersectia(a, b->next);
    // Liniile de mai jos sunt executate numai când: a->data ==
b->data
    struct Nod* temp = (struct Nod*)malloc(sizeof(struct Nod));
    temp->data = a->data;
    /* Avansăm ambele liste și le apelăm recursiv */
    temp->next = sortamIntersectia(a->next, b->next);
    return temp;
}
// Funcția de inserare a unui nod la începutul listei conectate
void push(struct Nod** head_ref, int new_data){
    // Alocăm nodul
      struct Nod* new_nod = (struct Nod*)malloc(sizeof(struct
Nod));
    // Introducem datele
    new_nod->data = new_data;
    // Anexăm vechea listă a noului nod
    new_nod->next = (*head_ref);
    // Mutăm capul pentru a indica noul nod
```

```
    (*head_ref) = new_nod;
}
// Funcția de imprimare a nodurilor într-o listă dată
void afiseazaLista(struct Nod* nod){
    while (nod != NULL) {
        cout << " "<<nod->data;
        nod = nod->next;
    }
}
// Programul principal care apelează funcțiile de mai sus
int main(){
    /* Începem cu listele goale */
    struct Nod* a = NULL; Nod* b = NULL;
    struct Nod* intersectie = NULL;
    /* Creăm prima listă sortată: 1-> 2-> 3-> 4-> 5-> 6 */
    push(&a, 6); push(&a, 5); push(&a, 4);
    push(&a, 3); push(&a, 2); push(&a, 1);
    /* Creăm a doua listă sortată: 0-> 1-> 2-> 4-> 6-> 8 */
    push(&b, 8); push(&b, 6); push(&b, 4);
    push(&b, 2); push(&b, 1); push(&b, 0);
    /* Găsim intersecția celor două liste */
    intersectie = sortamIntersectia(a, b);
    /* Afișăm elementelr celor două liste separat */
    cout<< "\n Lista A contine elementele: \t";
    afiseazaLista(a);
    cout<< "\n Lista B contine elementele: \t";
    afiseazaLista(b);
    /* Afișăm lista elementelr comune celor două liste */
     cout<< "\n\n Lista care contine elemente comune ale listei
A si B este:\n";
    afiseazaLista(intersectie); return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Lista A contine elementele:     1 2 3 4 5 6
Lista B contine elementele:     0 1 2 4 6 8

Lista care contine elemente comune ale listei A si B este:
1 2 4 6
```

Problema 5 – Inversarea unei liste simple

Fie avem o listă liniară simplu înlănțuită, imprimați inversul acesteia utilizând o funcție recursivă.

De exemplu, dacă lista liniară simplu înlănțuită dată este: 1-> 2-> 3-> 4, atunci ieșirea (afișarea elementelor în ordinea inversă) ar trebui să fie: 4-> 3-> 2-> 1.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
struct Nod {
    int data;
    struct Nod* next;
};
/* Funcție pentru a inversa elementele LSI */
void listaInvesa(Nod* head){
    // Cazul de bază
    if (head == NULL) return;
    // Tipărim lista după nodul principal
    listaInvesa(head->next);
    // Tipărim capul listei la sfârșit
    cout << head->data << " ";
}
/* Împingem un nod în lista, reținem că această funcție permite
să schimbăm capul listei.*/
void push(Nod** head_ref, char new_data){
    /* Alocăm nodul */
    Nod* new_nod = new Nod();
    /* Introducem datele */
    new_nod->data = new_data;
    /* Anexăm vechea listă a noului nod */
    new_nod->next = (*head_ref);
    (*head_ref) = new_nod;
}
/* Funcția de imprimare a nodurilor într-o listă dată */
void afiseazaLista(struct Nod* nod){
    while (nod != NULL) {
```

```
        cout <<nod->data<<" ";
        nod = nod->next;
    }
}
/* Programul principal care aplică funcțiile descrise mai sus.*/
int main(){
    // Creăm lista simplu înlănțuită: 1-> 2-> 3-> 4
    Nod* head = NULL;
    push(&head, 4); push(&head, 3);
    push(&head, 2); push(&head, 1);
    cout << "Elementele listei liniare simplu inlantuite sunt:\n\t";
    afiseazaLista(head); cout<<endl;
     cout << "Elementele listei liniare simplu inlantuite in odine inversa sunt:\n\t";
    listaInvesa(head);
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Elementele listei liniare simplu inlantuite sunt:
        1 2 3 4
Elementele listei liniare simplu inlantuite in odine inversa sunt:
        4 3 2 1
```

```
Elementele listei liniare simplu inlantuite sunt:
        16 27 38 49
Elementele listei liniare simplu inlantuite in odine inversa sunt:
        49 38 27 16
```

```
Elementele listei liniare simplu inlantuite sunt:
        57 95 32 41
Elementele listei liniare simplu inlantuite in odine inversa sunt:
        41 32 95 57
```

Problema 6 – Suma valorilor nodurilor egală cu K

Fie că avem trei liste liniare simplu înlănțuite, să notăm aceste liste cu: a, b și c. Găsiți un nod din fiecare listă astfel încât suma valorilor nodurilor să fie egală cu un număr dat introdus de la tastatură.

De exemplu, dacă cele trei liste au valorile: 12-> 6-> 29, 23-> 5-> 8 și 90-> 20-> 59, iar numărul dat este 101, atunci soluția va fi tripletul: „6 5 90”.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
struct Nod {
    int data;
    struct Nod* next;
};
/* Creăm o funcție pentru a insera un nod la începutul unei
liste */
void push (Nod** head_ref, int new_data){
    /* Alocăm nodul */
    Nod* new_nod = new Nod();
    /* Introducem datele */
    new_nod->data = new_data;
    /* Anexăm vechea listă a noului nod */
    new_nod->next = (*head_ref); (*head_ref) = new_nod;
}
/* Creăm o funcție pentru a verifica dacă există trei elemente
în a, b și c a căror sumă este egală cu numărul dat.
Funcția presupune că lista b este sortată în ordine crescătoare
și c este sortată în ordine descrescătoare. */
bool esteSortat(Nod *headA, Nod *headB, Nod *headC, int numar){
    Nod *a = headA;
    // Parcurgeți toate nodurile listei a
    while (a != NULL){
        Nod *b = headB; Nod *c = headC;
        // Pentru fiecare nod al listei a, împingem două noduri
din listele b și c
        while (b != NULL && c != NULL){
            // Dacă acesta este un triplet corespunzător sumei
```

```
date, imprimăm și returnăm valoarea adevărat.
            int sum = a->data + b->data + c->data;
            if (sum == numar){
            cout << "\nTripletul gasit este: "<< a->data << " "
<< b->data << " " << c->data<<endl;
            return true;
            }
      // Dacă suma acestui triplet este mai mică, căutăm valori
mai mari în b
            else if (sum < numar) b = b->next;
       // Dacă suma este mai mare, căutăm valori mai mici în c
             else c = c->next;
        }
        a = a->next; // Mergem înainte în lista a
    }
    cout << "Nu a fost gasit nici un triplet care sa corespunda
conditiei!";
    return false;
}
//Creăm funcția de afișare a elementelor unei liste oarecare
void afiseazaLista(struct Nod* nod){
    while (nod != NULL) {
        cout <<nod→data<<"\t"; nod = nod->next;
    }
}
/* Programul principal*/
int main(){
    int numar;
    /* Începem cu lista goală */
    Nod* headA = NULL; Nod* headB = NULL; Nod* headC = NULL;
    /*Cream LSI 'A': 10->15->5->20 */
    push (&headA, 20); push (&headA, 4);
    push (&headA, 15); push (&headA, 10);
    /*Cream LSI sortată 'B': 2->4->9->10 */
    push (&headB, 10); push (&headB, 9);
    push (&headB, 4); push (&headB, 2);
    /*Cream LSI sortată 'C': 8->4->2->1 */
    push (&headC, 1); push (&headC, 2);
    push (&headC, 4); push (&headC, 8);
    cout << "Introdu un numar natural: "; cin >> numar;
    // Afișarea tuturor elementelor fiecărei liste sub forma
liniară, tabulară
```

```
    cout << "Datele celor trei liste sunt: ";
    cout << "\n\t"; afiseazaLista(headA);
    cout << "\n\t"; afiseazaLista(headB);
    cout << "\n\t"; afiseazaLista(headC);
    esteSortat(headA, headB, headC, numar);
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Introdu un numar natural: 25
Datele celor trei liste sunt:
        10      15      4       20
        2       4       9       10
        8       4       2       1
Tripletul gasit este: 15 2 8
```

```
Introdu un numar natural: 52
Datele celor trei liste sunt:
        12      15      41      20
        20      43      59      60
        80      40      20      10
Tripletul gasit este: 12 20 20
```

```
Introdu un numar natural: 90
Datele celor trei liste sunt:
        12      15      41      20
        20      43      59      60
        80      40      20      10
Tripletul gasit este: 20 60 10
```

```
Introdu un numar natural: 45
Datele celor trei liste sunt:
        12      15      41      20
        20      43      59      60
        80      40      20      10
Tripletul gasit este: 15 20 10
```

2.5 Modele de probleme propuse

1. Elaborați subprograme în C++ pentru următoarele probleme simple.

- ***a.*** *concatenarea a două liste simplu înlănțuite;*
- ***b.*** *interschimbarea a două elemente impare consecutive din lista liniară simplu înlănțuită;*
- ***c.*** *sortarea ascendentă a unei liste simplu înlănțuite;*
- ***d.*** *găsirea elementului aflat pe poziția i de la sfârșitul listei.*
- ***e.*** *ștergerea a două elemente pare consecutive din lista liniară simplu înlănțuită.*
- ***f.*** *sortarea descendentă a unei liste simplu înlănțuite.*
- ***g.*** *găsirea elementului aflat pe poziția i de la începutul listei.*

2. Elaborați un program în C++ pentru următoarea problemă cu fișiere.

- ***a.*** *Să se creeze o listă liniară simplu înlănțuită care conține elemente numere reale citite dintr-un fișier text. Să se insereze între oricare două noduri din listă un nod care să conțină media aritmetică a celor două valori din cele două noduri.*
- ***b.*** *Să se creeze o listă liniară simplu înlănțuită care conține elemente numere întregi citite dintr-ul fisier text. Se citesc apoi două valori întregi x și y. Să se adauge după primul nod care conține valoarea x un nod care să conțină valoarea y.*

3. Elaborați un program în C++ pentru următoarele probleme.

3.1 *Să se realizeze un program pentru evidența abonaților telefonici dintr-o localitate. Pentru fiecare abonat se memorează numele, adresa, numărul de telefon. Programul trebuie să permită:*

- *adăugarea unui nou abonat;*
- *ștergerea unui abonat;*
- *modificarea datelor unui abonat;*

3.2 *Să se realizeze un program care permite utilizatorului să creeze şi să afişeze două mulţimi de numere naturale. Mulţimile se vor reprezenta sub forma unor liste simplu înlănţuite (un nod al listei conţine un element). Se vor dezvolta funcţii care permit:*

- *afişarea elementelor unei mulţimi de la dreapta la stânga;*
- *ştergerea unui element din a doua mulţime;*
- *afişarea elementelor care fac parte doar dintr-o mulţime.*

3.3 *Într-un depozit se află diverse utilaje. Informaţiile de interes pentru fiecare utilaj sunt: denumirea, anul de fabricaţie, sectorul de depozitare. Să se scrie un program care realizează următoarele:*

- *introducerea unui utilaj;*
- *afişarea utilajelor din depozit;*
- *mutarea unui utilaj dintr-un sector în altul;*
- *afişarea în ordine descrescătoare a vechimii, a utilajelor dintr-un sector citit de la tastatură.*

3.4 *Să se realizeze un program pentru evidenţa abonaţilor telefonici dintr-o localitate. Pentru fiecare abonat se memorează numele, adresa, numărul de telefon. Programul trebuie să permită:*

- *adăugarea unui nou abonat;*
- *ştergerea unui abonat;*
- *modificarea datelor unui abonat;*
- *afişarea numărului de telefon al unui abonat.*

3.5 *Să se realizeze un program pentru evidenţa abonaţilor de la biblioteca CEITI, din mun. Chișinău. Pentru fiecare abonat se memorează numele, adresa, numărul de telefon, grupa, nr. de cărți împrumutate. Programul trebuie să permită:*

- *adăugarea unui nou abonat;*
- *ștergerea unui abonat;*
- *modificarea grupei și a nr. de cărți împrumutate de la bibliotecă;*
- *afişarea numărului de telefon al unui abonat;*
- *afişarea tuturor cărților unui abonat după structura: autor, denumire carte, nr. de pagini, data împrumutării.*

4. Probleme suplimentare pentru LLSÎ

În exerciţiile din acest compartiment se va folosi o listă avînd ca informaţie utilă în nod un număr real:

```
typedef struct TNOD{
        float x; struct TNOD* next;
};
```

Nu se vor verifica posibilele erori la rezervarea spaţiului în heap, cauzate de lipsa acestuia. Pentru verificarea acestui tip de eroare se poate testa rezultatul întors de funcţia malloc (NULL dacă nu s-a putut face alocarea de memorie). Pentru uşurinţă în scriere, se va folosi următoarea macrodefiniţie:

```
#define NEW (TNOD*)malloc(sizeof(TNOD));
```

1. *Să se scrie programul pentru crearea unei liste simplu înlănţuite cu preluarea datelor de la tastatură. Sfîrşitul introducerii datelor este marcat standard. După creare, se va afişa conţinutul listei apoi se va elibera memoria ocupată.*
2. *Să se scrie funcţia pentru inserarea unui nod la începutul unei liste simplu înlănţuite. Funcţia are ca parametri capul listei în care se inserează şi valoarea care se inserează. Prin numele funcţiei se întoarce noul cap al listei.*
3. *Să se scrie funcţia pentru inserarea unui nod într-o listă simplu înlănţuită după un nod identificat prin valoarea unui cîmp. Dacă nodul căutat nu există, inserarea se face la sfîrşitul listei. Funcţia are ca parametri capul listei în care se inserează, valoarea care se inserează şi valoarea după care se inserează. Prin numele funcţiei se întoarce noul cap al listei.*
4. *Să se scrie funcţia pentru inserarea unui nod într-o listă simplu înlănţuită înaintea unui nod identificat prin valoarea unui cîmp. Dacă nodul căutat nu există, se face inserare la începutul listei. Funcţia are ca parametri capul listei în care se inserează, valoarea care se inserează şi valoarea înaintea cărei se inserează. Prin numele funcţiei se întoarce noul cap al listei.*

5. *Să se scrie funcţia pentru căutarea unui nod identificat prin valoarea unui cîmp într-o listă simplu înlănţuită. Funcţia are ca parametri capul listei în care se caută şi valoarea căutată. Prin numele funcţiei se întoarce adresa nodului care conţine informaţia căutată sau NULL, dacă informaţia nu a fost găsită în listă. Dacă sunt mai multe noduri cu aceeaşi valoare, se întoarce adresa ultimului.*
6. *Să se scrie funcţia pentru adăugarea unui nod la sfîrşitul unei liste simplu înlănţuite. Funcţia are ca parametri capul listei în care se inserează şi valoarea care se inserează. Prin numele funcţiei se întoarce noul cap al listei.*
7. *Să se scrie funcţia pentru ştergerea primului nod al unei liste simplu înlănţuite. Funcţia are ca parametru capul listei din care se şterge primul nod şi întoarce, prin numele ei, noul cap al listei.*
8. *Să se scrie funcţia pentru ştergerea unui nod identificat prin valoarea unui cîmp dintr-o listă simplu înlănţuită. Funcţia primeşte ca parametri adresa capului listei şi valoarea informaţiei utile din nodul care se şterge. Funcţia întoarce valoarea 1, dacă nu a găsit nodul căutat, sau 0, dacă a efectuat ştergerea.*
9. *Să se scrie funcţia pentru ştergerea ultimului nod dintr-o listă simplu înlănţuită. Funcţia are ca parametru capul listei din care se şterge ultimul nod şi întoarce, prin numele ei, noul cap al listei.*
10. *Să se scrie funcţia pentru ştergerea unei liste simplu înlănţuite. Funcţia are ca parametru capul listei care trebuie ştearsă şi întoarce, prin numele ei, noul cap al listei (valoarea NULL).*
11. *Să se scrie funcţia pentru ştergerea nodului aflat după un nod identificat prin valoarea unui cîmp. Funcţia are ca parametri capul listei şi valoarea nodului căutat (după care se şterge). Prin numele funcţiei se întoarce valoarea 0, dacă s-a făcut ştergerea, sau 1, dacă nodul căutat nu a fost găsit.*
12. *Să se scrie funcţia pentru ştergerea nodului aflat înaintea nodului identificat prin valoarea unui câmp. Funţia are ca parametri capul listei, valoarea din nodul căutat (înaintea căruia se face ştergerea) şi adresa unde se înscrie parametrul de eroare (0 dacă se face ştergerea, 1 dacă nodul căutat este primul, 2 dacă nodul căutat nu a fost găsit). Funcţia întoarce noul cap al listei.*

2.6 Portofoliul elevului pentru LLSÎ

Se consideră o listă liniară simplu înlănțuită care memorează valori întregi. Elaborați subprograme pentru următoarele operațiuni:

a. Introduceți un număr nou în listă;
b. Afișați elementele pare și impare ale listei;
c. Afișați elementele pare pozitive și pare negative ale listei;
d. Afișați elementele impare pozitive și impare negative ale listei;
e. Afișați elementele în ordine crescătoare ale listei;
f. Afișați elementele în ordine descrescătoare ale listei;
g. Să se șteargă primul nod (ultimul nod)care conține valoarea x.
h. Să se șteargă nodurile care conțin pătrate perfecte.
i. Să se șteargă nodurile care conțin numere Fibonacci.

Fiecare elev din subgrupă va realiza o bibliotecă ce va include operațiile de mai sus, cu condiția că elementele inițiale ale listei sunt citite din fișier, fiecare elev va avea o listă individuală conform numărului de ordine.

Nr.	Secvența de numere	Nr.	Secvența de numere
1	50 31 30 35 82 85 16 18 58 61	**9**	21 37 56 32 44 71 99 87 72 90
2	79 59 45 14 57 23 73 83 51 62	**10**	19 12 49 18 37 71 39 68 42 98
3	77 81 66 84 95 43 38 65 15 76	**11**	75 46 97 40 57 70 54 73 14 59
4	26 64 39 13 17 10 40 55 29 48	**12**	56 31 44 27 96 80 81 86 58 65
5	47 74 27 91 75 88 33 25 93 42	**13**	53 21 50 67 83 69 93 32 43 17
6	67 22 69 96 46 97 28 12 41 36	**14**	41 95 89 77 29 33 87 51 99 72
7	94 63 92 34 53 78 11 52 19 86	**15**	61 76 45 24 79 85 52 92 15 36
8	70 24 89 80 60 49 68 20 98 54	**16**	82 64 74 38 23 16 63 30 10 62

2.7 Model de test grilă pentru LLSÎ

Nr.	Conținutul	Punctaj
1	*Lista liniară simplu înlănțuită este parcursă pornind de la pointerul spre primul element și avansând folosind pointerii din structură până la pointerul NULL.* ◯ adevărat ◯ fals	*2p*
2	*Fiecare nod al listei liniare simplu înlănțuite conține, în afară de informația utilă, adresa precedentului element.* ◯ adevărat ◯ fals	*2p*
3	*O listă liniară simplu înlănțuită conține elemente (noduri) a căror valori constau din două părți: informația utilă și informația de legătură.* ◯ adevărat ◯ fals	*2p*
4	*Operația de . . . a unui element dintr-o listă presupune parcurgerea listei pentru identificarea nodului în funcție de un criteriu.* ◯ inserare ◯ căutare ◯ ștergere ◯ interschimbare	*2p*
5	*Listele liniare simplu înlănțuite sunt structuri dinamice . . . spre deosebire de masive.* ◯ omogene ◯ neomogene ◯ eterogene ◯ exogene	*2p*
6	*Listele liniare simplu înlănțuite sunt formate din noduri care sunt variabile de tip . . .* ◯ pointer ◯ struct ◯ array ◯ string	*2p*
7	*Un element poate fi căutat în lista liniară simplu înlănțuită în următoarele cazuri:* ☐ după poziție ☐ dacă lista este vidă ☐ după valoare ☐ alt răspuns	*2p*

8	*Un element poate fi inserat în lista liniară simplu înlănțuită în următoarele cazuri:* ☐ la început ☐ la sfârșit ☐ după un elemen ☐ alt răspuns	**3p**
9	*Un element poate fi șters din lista liniară simplu înlănțuită în următoarele moduri:* ☐ din interior (diferit de capul listei) ☐ după o anumită valoare ☐ de pe o anumită poziție ☐ răspunsul corect lipsește	**3p**

10	*Fie următoarea secvență de cod în limbajul C++:*
	
	`void Inserare(Element* &cap, Date val){`
	`  Element *elem = new Element;`
	`  elem->valoare = val;`
	`  elem->urmator = NULL;`
	`  if (cap == NULL) cap = elem;`
	`  else {`
	`  Element *nod = cap;`
	`  while (nod->urmator != NULL)`
	`  nod = nod->urmator;`
	`  nod->urmator = elem;`
	`  }`
	`}`
	
	Studiați atent subprogramul și stabiliți care ar fi tipul de inserare prezentat.
	◯ la început ◯ la sfârșit ◯ după un element
		5p

Barem de notare propus

Nota	10	9	8	7	6	5	4	3	2
%	100-95	94-88	87-78	77-63	62-48	47-33	32-21	20-10	9-5
Punctaj	25-24	23-22	21-20	19-16	15-12	11-8	7-5	4-3	2-1

3. LISTE DUBLU ÎNLĂNȚUITE

3.1 Introducere

O componentă a unei liste dublu înlănțuite se declară ca o dată structurată de tip înregistrare, formată din trei câmpuri: informația propriu-zisă (care poate fi de orice tip: numeric, caracter, pointer, tablou, înregistrare) și informațiile de legătură (adresa la care e memorată următoarea componentă și adresa la care e memorată precedenta componentă). Ultima componentă va avea informația de legătură corespunzătoare următoarei adrese NULL (sau 0), cu semnificația ca după ea nu mai urmează nimic (reține adresa „nici o adresă" a următoarei componente). La fel și în cazul primei componente pentru câmpul adresa precedentă.

Listele dublu înlănțuite sunt structuri de date dinamice omogene. Ele au aceleași caracteristici de bază ca și listele simplu înlănțuite. Diferența față de acestea constă în faptul că, pentru fiecare nod, se reține și adresa elementului anterior, ceea ce permite traversarea listei în ambele direcții.

Lista dublu înlănțuită poate fi reprezentată grafic astfel:

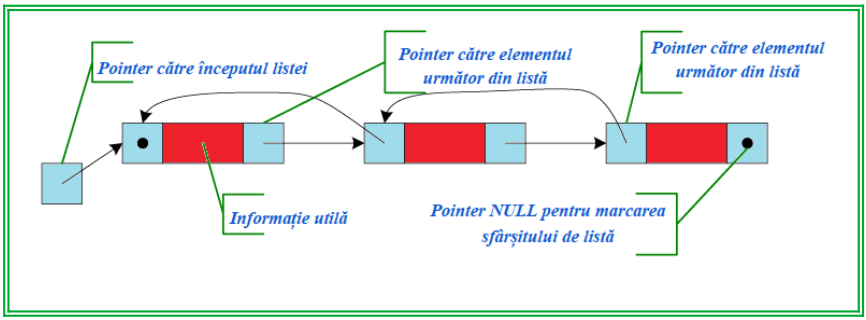


Figura 2.1 – Reprezentarea grafică a unei liste liniare dublu înlănțuite

3.2 Structura unei liste dublu înlănțuite

Conform celor enunțate anterior memorarea și prelucrarea acestor structuri de date este similară cu cea a listelor liniare simplu înlănțuite, ba chiar unele prelucrări (cum ar fi ștergerea și inserarea înainte se simplifica având într-un mod facil acces la componenta anterioară).

Structura folosită este similară cu cea de la liste simple. Pentru a asigura un grad mai mare de generalitate a listei, a fost creat un alias pentru datele utile (în cazul nostru un întreg):

```
// Datele asociate unui element (nod) dintr-o listă
typedef int Date;
```

În cazul în care se dorește memorarea unui alt tip de date, trebuie schimbată doar declarația aliasului Date. Pentru memorarea listei se foloseste o structura autoreferita. Această structură va avea forma:

```
struct Element{
   Date valoare; // datele efective memorate
   Element* next, *back; // legatura către nodul următor
};
```

*În cazul în care elemenul este ultimul din listă, pointerul **next** va avea valoarea NULL. Pentru primul element, pointerul **back** va avea valoarea NULL. Declararea listei se face sub forma:*

```
// Declarare listă vidă
Element* cap = NULL;
```

*În cazul în care se dorește crearea unei liste circulare, pointerul **next** al ultimului element va referi primul element al listei, iar pointerul **back** al primului element va referi ultimul element.*

3.3 Operații principale efectuate

3.3.1 Parcurgerea și afișarea listei

Lista este parcursă pornind de la pointer spre primul element și avansând folosind pointerii din structură până la sfârșitul listei (pointer NULL).

```
// Parcurgere și afișare pentru o listă dublă
void Afisare(Element* cap) {
     // Cât timp mai avem elemente în listă
     while (cap != NULL) {
          // Afișează elementul curent
          cout << cap->valoare << endl;
          // Avansează la elementul următor
          cap = cap -> next;
     }
}
```

Pentru parcurgerea inversă a listei se va porni cu un pointer către ultimul element al listei, iar avansarea se va face folosind secvența:

```
cap = cap->back;
```

3.3.2 Inserarea (adăugarea) unui element în listă

Pentru inserarea unui element în listă, există câteva modalități:

- *inserarea la începutul listei;*
- *inserarea la sfârșitul listei;*
- *inserarea după un element dat.*

Inserarea la începutul listei este cazul cel mai simplu: trebuie doar alocat elementul, legat de primul element din listă și repoziționarea capului listei.

```
void InserareInceput(Element* &cap, Date val) {
    // Alocare nod și inițializare valoare
    Element *elem = new Element; elem->valoare = val;
    elem->back = NULL;
```

```
    // Legare nod în listă
    elem->next = cap;
    if (cap != NULL) cap-> back = elem;
    // Mutarea capului listei
    cap = elem;
}
```

Inserarea la sfârșitul listei este cazul în care trebuie mai întâi parcursă lista și după aceea adăugat elementul și legat de restul listei.

```
void InserareSfarsit(Element* &cap, Date val){
    // Alocare și inițializare nod
    Element *elem = new Element; elem->valoare = val;
    elem->next = NULL; elem->back = NULL;
    // Dacă avem lista vidă doar modificăm capul listei
    if (cap == NULL) cap = elem;
    else {
       // Parcurgem lista până la ultimul nod
       Element *nod = cap;
       while (nod->next != NULL) nod = nod -> next;
       // Adăugăm elementul nou în listă
       nod->next = elem; elem->back = nod ;
    }
}
```

Inserarea unui elemen după un element dat (stabilit) va fi reprezentată de următoarea secvență de cod:

```
void InserareInterior(Element* &cap, Element* p, Date val){
    // Alocare și inițializare nod
    Element *elem = new Element; elem->valoare = val;
    elem->next = NULL; elem->back = NULL;
    // Lista vidă
    if (cap == NULL){
       cap = ele m; return;
    }
    // Inserare la începutul listei
    if (cap == p){
       elem->next = cap; cap->back = elem; cap = elem;
       return;
```

```
    }
    // Inserare în interior
    elem->next = p->next; elem->back = p;
    p->next->back = elem; p->next = elem;
}
```

3.3.3 Căutarea unui element în listă

Căutarea unui element dintr-o listă presupune parcurgerea listei pentru identificarea nodului în funcție de un criteriu. Cele mai uzuale criterii sunt cele legate de poziția în cadrul listei și de informațiile utile conținute de nod.

Rezultatul operației este adresa primului element găsit sau NULL. Pentru căutarea unui element în listă, există câteva modalități:

- *căutarea după poziție;*
- *căutarea după valoare.*

Pentru căutarea unui element după poziție, se avansează pointerul cu numărul de poziții specificat:

```
Element* CautarePozitie(Element* cap, int pozitie) {
     int i = 0; // Poziția curentă
     // Parcurge lista până la poziția cerută sau
     // până la sfârșitul listei
     while (cap != NULL && i < pozitie) {
           cap = cap -> next; i++;
     }
     // Dacă lista conține elementul
     if (i == poziție) return cap;
     else return NULL;
}
```

Pentru căutarea unui element după valoare, se parcurge lista până la epuizarea acesteia sau identificarea elementului:

```
Element* CautareValoare(Element* cap, Date val) {
     // Parcurge lista până la găsirea elementului sau
     // epuizarea listei
     while (cap != NULL && cap->valoare != val)
     cap = cap -> next;
```

```
    return cap;
}
```

3.3.4 Ștergerea unui element din listă

Pentru ștergerea unui element din listă, există câteva modalități:

- *ștergerea unui element din interiorul listei;*
- *ștergerea unui element de pe o anumită poziție;*
- *ștergerea după o valoare.*

Pentru ștergerea unui element din interiorul listei (diferit de capul listei) prezentăm următoarea secvență de cod:

```
void StergereElementInterior(Element* elem) {
   //  Scurcircuităm elementul
   elem->back->next = elem->next;
   elem->next-> back = elem->back;
   // Ștergem elementul
   delete elem;
}
```

Pentru ștergerea unui element de pe o anumită poziție trebuie să ținem cont de faptul că dacă elementul este primul din listă, atunci se modifică capul listei, altfel se caută elementul și se șterge folosind funcția definită anterior.

```
void StergerePozitie(Element* &cap, int pozitie) {
// Dacă lista e vidă, nu facem nimic
if (cap == NULL) return;
// Dacă este primul, atunci îl ștergem și mutăm capul
if (pozitie == 0) {
    Element* deSters = cap; cap = cap -> next;
    cap->back = NULL; delete deSters; return;
}
// Dacă este în interor, atunci folosim funcția de ștergere
Element* elem = CautarePozitie(cap, pozitie);
StergereElementInterior(elem);
}
```

Pentru ștergerea unui element după o valoare, trebuie să ținem cont de faptul că se caută elementul și se folosește funcția de ștergere element.

```
void StergereValoare(Element* &cap, Date val) {
// Dacă lista e vidă, nu facem nimic
if (cap == NULL) return;
// Dacă este primul, atunci îl ștergem și mutăm capul
if (cap->valoare == val) {
   Element* deSters = cap; cap = cap -> next;
   cap->back = NULL; delete deSters; return;
}
// Căutăm elementul
Element* elem = CautareValoare(cap, val);
// Dacă a fost găsit, atunci îl ștergem
if (elem->next != NULL) StergereElementInterior(elem);
}
```

Remarcă:

- *Dacă ați reușit să realizați un exemplu de fișier antet pentru structura dinamică studiată în compartimentul precedent, creați un asemenea fișier pentru structura dinamică respectivă, incluzând operațiile studiate în acest compartiment.*
- *Dacă ați uitat cum se poate de creat un asemenea fișier în C++ este necesar să analizați pașii prezentați în exemplele din anexa 1.*
- *De asemenea la dorință se poate de îmbunătățit acest fișier antet cu alte subprograme pe care le veți aplica la rezolvarea problemelor.*

3.4 Modele de probleme rezolvate

Problema 1 – Inversarea listei duble

Fie avem o listă dublu înlănțuită. Să se inverseze elementele acestei liste.

De exemplu, dacă elementele listei sunt: 7->8->12->10->5->3->9, atunci lista inversă ar trebui să fie 9->3->5->10->12-> 8 ->7.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
// Nodul unei liste dublu înlănțuite
struct Nod {
   int data;
   struct Nod *prev, *next;
};
// Crearea și returnarea unui nou nod al unei liste duble
Nod* newNod(int val){
   Nod* temp = new Nod;
   temp->data = val;
   temp->prev = temp->next = nullptr;
   return temp;
}
void afisareLista(Nod* capul){
   while (capul->next != nullptr) {
      cout << capul->data << " <==> ";
      capul = capul->next;
   }
   cout << capul->data << endl;
}
// Introducem un nou nod în capul listei
void Introducem(Nod** capul, int nod_data){
   Nod* temp = newNod(nod_data);
   temp->next = *capul; (*capul)->prev = temp;
   (*capul) = temp;
}
// Funcția de inversare a unei liste duble
```

```
void inversareLista(Nod** capul){
   Nod* left = *capul, * right = *capul;
   // Parcurgem întreaga listă și setăm spre dreapta
   while (right->next != nullptr)
   right = right->next;
   // Schimbăm datele la stânga și la dreapta deplasându-le
unul spre celălalt până când se întâlnesc sau se încrucișează
   while (left != right && left->prev != right){
      // Schimbăm datele indicatorului la stânga și la dreapta
      swap(left->data, right->data);
      left = left->next; // Indicatorul stâng il avansăm
      right = right->prev; // Indicatorul drept il avansăm
   }
}
// Programul principal
int main(){
   Nod* headNod = newNod(9);
   Introducem(&headNod, 3); Introducem(&headNod, 5);
   Introducem(&headNod, 10); Introducem(&headNod, 12);
   Introducem(&headNod, 8); Introducem(&headNod, 7);
   cout<<"\nDatele initiale pentru lista dublu inlantuita:\n\
t";
   afisareLista(headNod);
   cout<<"\nDatele inversate pentru lista dublu inlantuita:\n\
t";
   inversareLista(&headNod); afisareLista(headNod);
   return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Datele initiale pentru lista dublu inlantuita:
        7 <==> 8 <==> 12 <==> 10 <==> 5 <==> 3 <==> 9

Datele inversate pentru lista dublu inlantuita:
        9 <==> 3 <==> 5 <==> 10 <==> 12 <==> 8 <==> 7
```

```
Datele initiale pentru lista dublu inlantuita:
        7 <==> 8 <==> 9 <==> 10 <==> 11 <==> 12 <==> 13

Datele inversate pentru lista dublu inlantuita:
        13 <==> 12 <==> 11 <==> 10 <==> 9 <==> 8 <==> 7
```

Problema 2 – Inserarea elemente înainte și după

Fie avem o listă dublu înlănțuită goală. Să se elaboreze subprograme pentru a adăuga un element în listă pe poziția din față (prima poziție), pe poziția din spate (ultima poziție). Să se verifice dacă un careva element introdus există în listă, dacă există, atunci să se șteargă acel element și să se afișeze lista. Se propune de a se aplica conceptul de POO.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
//Creăm un template pentru clasa ListaDubla
template<class T>
class ListaDubla{
    struct Nod{
        T data;
        Nod* next; Nod* prev;
        //nullptr este un cuvânt cheie care poate fi folosit în
toate locurile în care este așteptat NULL
        Nod(T val): data(val), next(nullptr), prev(nullptr) {}
    };
    Nod *cap, *coada;
    public:
      ListaDubla(): cap(nullptr), coada(nullptr) {}
     ~ListaDubla(){ //destructor
          Nod *tmp = nullptr;
          while (cap){
              tmp = cap; cap = cap->next;
              delete tmp;
          }
          cap = nullptr;
      }
    ListaDubla(const ListaDubla<T> & dll) = delete;
    ListaDubla& operator=(ListaDubla const&) = delete;
    // Inserarea unui nou element pe poziția din față
    void insereareInainte(T val){
        Nod *nod = new Nod(val); Nod *tmp = cap;
        if (cap == nullptr){
            cap = nod; coada = nod;
```

```
        }
        else {
            nod->next = cap; cap = nod; nod->next->prev = nod;
        }
    }
    // Inserarea unui nou element pe poziția din spate
    void inserareUrmator(T val){
        Nod *nod = new Nod(val);
        if(coada->next == nullptr){
            coada->next = nod; coada = nod;
        }
    }
    // Ștergerea unui nou element din lista dublă
    void stergeVal(T val){
        Nod* find = cautaVal(val); Nod *tmp = cap;
        if(tmp == find){
            cap = tmp->next;
        }
        else{
            while(find != nullptr){
                if(tmp->next == find){
                    tmp->next=find->next;
                    find->next->prev=tmp; delete find; return;
                }
                tmp = tmp->next;
            }
        }
    }
    //Creăm un template pentru verifica prezența unui anumit
element în lista dubla
    template <class U>
    friend ostream & operator<<(ostream & os, const
ListaDubla<U> & dll){
        dll.afisare(os); return os;
    }
    private:
        Nod *cautaVal(T n){ //returnează nodul numărului dat
            Nod *nod = cap;
            while(nod != nullptr){
                if(nod->data == n) return nod;
                nod = nod->next;
            }
```

```
            cerr << "\tNu exista acest element in lista! \n";
            return nullptr;
        }
    //Creăm un template pentru clasa ListaDubla
    void afisare(ostream& out = cout) const{
        Nod *nod = cap;
        while(nod != nullptr){
            out << nod->data << " "; nod = nod->next;
        }
    }
};
// Programul principal
int main(){
  ListaDubla<int> l1;  cout<<"Lista initiala: "<<l1<<"\n";
  l1.inserareInainte(3); cout<<"Lista dupa inserarea inainte: \
t"<<l1<<"\n";
  l1.inserareUrmator(5); cout<<"Lista dupa inserarea in
spate: \t"<<l1<<"\n";
  l1.inserareUrmator(12); cout<<"Lista dupa inserarea in spate:
\t"<<l1<<"\n";
  l1.inserareInainte(6); cout<<"Lista dupa inserarea inainte: \
t"<<l1<<"\n";
  l1.inserareUrmator(88); cout<<"Lista dupa inserarea in spate:
\t"<<l1<<"\n";
  l1.stergeVal(12); cout<<"Lista dupa stergerea unui nod: \
t"<<l1<<"\n";
  cout<<"Dorim sa stergem un nod inexistent: \n";
l1.stergeVal(11); return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Lista initiala:
Lista dupa inserarea inainte:   3
Lista dupa inserarea in spate:  3 5
Lista dupa inserarea in spate:  3 5 12
Lista dupa inserarea inainte:   6 3 5 12
Lista dupa inserarea in spate:  6 3 5 12 88
Lista dupa stergerea unui nod:  6 3 5 88
Dorim sa stergem un nod inexistent:
       Nu exista un acest element in lista!
```

Problema 3 – Permutările circulare ale șirului de elemente

Să se creeze o listă liniară dublu înlănțuită care să memoreze valori întregi prin adăugare la sfârșitul listei. Să se scrie o funcție care primește ca parametru adresa primului nod al listei și mută primul nod după ultimul. Afișați într-un fișier toate permutarile circulare ale șirului de numere memorat în listă.

Prezentarea soluției problemei în limbajul C++

```
#include <fstream>
#include <iostream>
using namespace std;
ifstream f("date.in"); // Anexarea fișierelor de intrare
ofstream g("date.out"); // Anexarea fișierelor de ieșire
// Structura nodului unei liste duble
struct nod{
    int info; nod *prec,*urm;
}; nod *prim;
// Funcția care adaugă cate un element în lista dublă
void adaug(nod *&prim,int x){
    nod *u=prim;
    if(prim){
        while(u->urm)
            u=u->urm; nod *nou=new nod; nou->info=x;
            u->urm=nou; nou->prec=u; nou->urm=0;
    }
    else {
        prim=new nod; prim->info=x; prim->urm=0; prim->prec=0;
    }
}
// Funcția care crează lista dublă după datele din fișier
void creare(nod *&prim){
    int x;
    while(f>>x) adaug(prim,x);
}
// Funcția care afișează lista dublă în fișier
void afisare(nod *prim){
    nod *p=prim;
    while(p->urm){
```

```
        g<<p->info<<" "; p=p->urm;
    }
    g<<p->info<<" "; g<<endl;
}
// Funcția care crează rotația elementelor din lista dublă spre
dreapta
void rotareLista(nod *&prim){
    nod *u=prim;
    while(u->urm)
        u=u->urm; u->urm=prim; prim->prec=u; prim=prim->urm;
        prim->prec=0; u=u->urm; u->urm=0;
}
int numar(nod *prim){
    nod *p; int k=0; p=prim;
    while(p){
        p=p->urm; k++;
    } return k;
}
// Programul principal
int main(){
    g<<"Am afisat elementele initiale ale listei duble!\n";
    creare(prim); afisare(prim);
    g<<"Afisam elementele listei duble dupa fiecare rotatie!\
n";
    int n=numar(prim);
    for(int i=1;i<=n;i++){
        g<<"Pasul "<<i<<": "; rotareLista(prim); afisare(prim);
    }
    g<<"Afisam efectuat rotatia elementelor listei duble pana
am revenit la datele initiale!\n";
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Am afisat elementele initiale ale listei duble!
10 20 30
Afisam elementele listei duble dupa fiecare rotatie!
Pasul 1: 20 30 10
Pasul 2: 30 10 20
Pasul 3: 10 20 30
Am efectuat rotatia pana am revenit la datele initiale!
```

Problema 4 – Ultimul nod în fața primului

Să se creeze o listă liniară dublu înlănțuită care să memoreze valori întregi prin adăugare la sfârșitul listei. Să se scrie o functie care primește ca parametru adresa primului nod al listei și mută ultimul nod în fața primului. Afișați într-un fișier rezultatul obținut.

Prezentarea soluției problemei în limbajul C++

```
#include <fstream>
#include <iostream>
using namespace std;
ifstream f("date.in"); // Anexarea fișierelor de intrare
ofstream g("date.out"); // Anexarea fișierelor de ieșire
// Structura nodului unei liste duble
struct nod{
    int info; nod *prec,*urm;
}; nod *prim;
// Funcția care adaugă cate un element în lista dublă
void adaug(nod *&prim,int x){
    nod *u=prim;
    if(prim){
        while(u->urm)
            u=u->urm; nod *nou=new nod; nou->info=x;
            u->urm=nou; nou->prec=u; nou->urm=0;
    }
    else {
        prim=new nod; prim->info=x; prim->urm=0; prim->prec=0;
    }
}
// Funcția care crează lista dublă după datele din fișier
void creare(nod *&prim){
    int x;
    while(f>>x) adaug(prim,x);
}
// Funcția care afișează lista dublă în fișier
void afisare(nod *prim){
    nod *p=prim;
    while(p->urm){
        g<<p->info<<" "; p=p->urm;
```

```
    }
    g<<p->info<<" "; g<<endl;
}
// Funcția care primeste ca parametru adresa primului nod al
listei si muta ultimul nod in fata primului
void miscare(nod *&prim){
    nod *u=prim;
    while(u->urm) u=u->urm;
        u->prec->urm=0; u->prec=0; u->urm=prim;
        prim->prec=u; prim=u;
 }
// Programul principal
int main(){
    g<<"\nAm afisat elementele initiale ale listei duble!\n";
    creare(prim); afisare(prim);
    g<<"\nAfisam elementele listei duble dupa modificarea efec-
tuata!\n";
    miscare(prim); afisare(prim);
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Am afisat elementele initiale ale listei duble!
1 2 3 4 5 6 7 8 9

Afisam elementele listei duble dupa modificarea efectuata!
9 1 2 3 4 5 6 7 8
```

```
Am afisat elementele initiale ale listei duble!
9 8 7 6 5 4 3 2 1

Afisam elementele listei duble dupa modificarea efectuata!
1 9 8 7 6 5 4 3 2
```

```
Am afisat elementele initiale ale listei duble!
1 9 3 7 6 4 2 5 0 8

Afisam elementele listei duble dupa modificarea efectuata!
8 1 9 3 7 6 4 2 5 0
```

Problema 5 – Puncte în spațiu

Să se ceeze o listă dublu înlănțuită cu coordonate ale mai multor puncte din spatiu și apoi să se afișeze pe ecran punctele cu coordonatele lor de la început spre sfârșit și invers.

Prezentarea soluției problemei în limbajul C++

```
#include<iostream>
#include<stdlib.h>
#include<conio.h>
using namespace std;
///************* DECLARAREA VARIABILELOR ***************
    float wx,wy,wz;
    char raspuns;
    int n=0,nf;
///************** STRUCTURA PUNCT SPAȚIAL **************
struct tip_punct{
    float x,y,z;
    struct tip_punct *back;
    struct tip_punct *next;
} *var_punct,*primul_punct,*ultimul_punct;
///*************** AFIȘAREA ÎN ORDINEA INTRODUSĂ *******
void afisare(){
cout<<"\n Lista punctelor de la inceput pana la sfarsit:";
cout<<"\n ============================================";
var_punct=primul_punct; n=0;
while(var_punct->next!=NULL){
cout<<"\n   P"<<n<<"   ("<<var_punct->x<<",   "<<var_punct->x<<",
"<<var_punct->x<<") ";
var_punct=var_punct->next; n++;
}
cout<<"\n   P"<<n<<"   ("<<var_punct->x<<",   "<<var_punct->x<<",
"<<var_punct->x<<") ";
cout<<"\n ============================================";
}
///*************** AFIȘAREA ÎN ORDINEA INVERSĂ ********
void afisareInversa(){
nf=n;
cout<<"\n Lista punctelor de la sfarsit catre inceput:";
```

```
cout<<"\n ==========================================";
var_punct=ultimul_punct;
while(var_punct->back!=NULL){
cout<<"\n  P"<<nf<<"  ("<<var_punct->x<<",  "<<var_punct->x<<",
"<<var_punct->x<<") ";
var_punct=var_punct->back; nf--;
}
cout<<"\n  P"<<nf<<"  ("<<var_punct->x<<",  "<<var_punct->x<<",
"<<var_punct->x<<") ";
cout<<"\n ==========================================";
}
///***************** PROGRAMULPRINCIPAL ***************
int main(){
    ///Citirea coordonatelor primului punct
          primul_punct=(struct  tip_punct*)malloc(sizeof(struct
tip_punct));
    if(primul_punct==NULL){
         cout<<"\n Memorie insuficienta pentru primul punct!";
return 0;
    }
    cout<<"Abscisa punctului P("<<n<<"), x"<<n<<"=\t";
    cin>>wx;
    cout<<"Ordonata punctului P("<<n<<"), y"<<n<<"=\t";
    cin>>wy;
    cout<<"Cota punctului P("<<n<<"), z"<<n<<"=\t";
    cin>>wz;
    primul_punct->x=wx; primul_punct->y=wy;
    primul_punct->z=wz;
    primul_punct->back=NULL; primul_punct->next=NULL;
    ultimul_punct=primul_punct; var_punct=primul_punct;
    cout<<"\n Continuati adaugarea de puncte?(d=Da/n=Nu):\t";
    raspuns=getche(); system("CLS");
    while((raspuns=='D')||(raspuns=='d')){
        /// Citirea coordonatelor unui alt punct (urmator)
        n++;
        cout<<"\nAbscisa punctului P("<<n<<"), x"<<n<<"=\t";
        cin>>wx;
        cout<<"Ordonata punctului P("<<n<<"), y"<<n<<"=\t";
        cin>>wy;
        cout<<"Cota punctului P("<<n<<"), z"<<n<<"=\t";
        cin>>wz;
          ultimul_punct=(struct tip_punct*)malloc(sizeof(struct
```

```
tip_punct));
        if(ultimul_punct==NULL){
                cout<<"\n Memorie insuficienta pentru ultimul
punct!"; return 0;
        }
        ultimul_punct->x=wx; ultimul_punct->y=wy;
        ultimul_punct->z=wz; ultimul_punct->back=var_punct;
        ultimul_punct->next=NULL;
        var_punct->next=ultimul_punct;
        var_punct=ultimul_punct;
         cout<<"\n Continuati adaugarea de puncte?(d=Da/n=Nu):\
t";
        raspuns=getche(); system("CLS");
    }
    /// Listarea punctelor de la început până la sfârșit
    afisare();
    /// Listarea punctelor de la sfârșit către început
    afisareInversa();
}
```

Variabilele alocate dinamic vor fi memorate în zona Heap. La finalul execuției programului se poate cunoaște adresa primului, respectiv ultimului punct memorat. Locurile din memorie ale celorlalte puncte sunt necunoscute.

*Parcurgerea si extragerea datelor referitoare la puncte sunt permise numai secvențial, cu ajutorul variabilelor de tip pointer incluse în variabilele dinamice (**back**=anterior, **next**=următor).*

*Dacă în variabila de tip pointer **var_punct** se pune conținutul variabilei **primul_punct**, atunci **var_punct = primul_punct** și se poate referi primul punct prin **var_punct**. Referirea la un câmp al variabilei dinamice de tip punct se face prin **var_punct->** urmat de numele câmpului:*

- ***var_punct->x** - conține abscisa primului punct;*
- ***var_punct->y** - conține ordonata primului punct;*
- ***var_punct->z** - conține cota primului punct;*
- ***var_punct->next** - conține adresa unde este memorat următorul punct.*

*De asemenea se poate iniția un șir de instrucțiuni pentru a extrage pe rând datele referitoare la punctele memorate astfel (**var_punct** va conține adresa de memorie în care a fost memorat următorul punct): **var_punct = var_punct->next;***

*Referirea la câmpurile noului punct se va face la fel ca mai înainte. Se va continua această procedură de prelucrare a datelor referitoare la punctele consecutive până când variabila **var_punct** va avea valoarea egala cu variabila **ultimul_punct**, sau când câmpul **var_punct->next** va avea valoarea NULL. Se observă că variabilele dinamice (de tip pointer) pot interveni în program în două moduri:*

- *Primul, când se folosește variabila **simplu**, fără referire, caz în care singurele operații permise între ele sunt cele de atribuire (inițializarea unei variabile dinamice cu constanta NULL sau tansferul conținutului între variabile dinamice de același tip).*
- *Al doilea mod este când se folosește ca și referire la o variabilă dinamică (**var_dinamica1-> var_dinamica2)**, caz în care se lucrează cu variabile dinamice.*

Spre exemplu dacă avem coordonatele a 4 puncte din spațiu: P0 (1,1,1), P1 (2,2,2), P2 (3,3,3) și P3 (4,4,4). După execuție vom avea:

```
Lista punctelor de la inceput pana la sfarsit:
==============================================
P0 (1, 1, 1)
P1 (2, 2, 2)
P2 (3, 3, 3)
P3 (4, 4, 4)
==============================================
Lista punctelor de la sfarsit catre inceput:
==============================================
P3 (4, 4, 4)
P2 (3, 3, 3)
P1 (2, 2, 2)
P0 (1, 1, 1)
==============================================
```

Problema 6 – Metoda InsertionSort

Fie avem o listă dublu înlănțuită. Să se sorteze elementele acestei liste folosind tehnica de sortare prin inserție.

De exemplu, dacă elementele din listă sunt: 8-> 12-> 10-> 5-> 3-> 9, atunci lista sortată conform tehnicii stabilite ar trebui să fie 3-> 5-> 8-> 9-> 10-> 12.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
// Nodul unei liste dublu înlănțuite
struct Nod {
    int data;
    struct Nod* prev, *next;
};
// Crearea și returnarea unui nou nod al unei liste duble
struct Nod* getNod(int data){
    // Alocăm nodul
    struct Nod* newNod = (struct Nod*)malloc(sizeof(struct
Nod));
    // Introducem datele
    newNod->data=data; newNod->prev = newNod->next = NULL;
    return newNod;
}
// Funcția pentru a insera un nou nod în modul de sortare într-
o listă sortată dublu înlănțuită
void InsertionSort(struct Nod** referintaCap, struct Nod*
newNod){
    struct Nod* curent;
    // Dacă lista este goală
    if (*referintaCap == NULL) *referintaCap = newNod;
    // Dacă nodul urmează să fie inserat la începutul listei
dublu înlănțuită
    else if ((*referintaCap)->data >= newNod->data) {
        newNod->next = *referintaCap;
        newNod->next->prev = newNod; *referintaCap = newNod;
    }
```

```
    else {
        curent = *referintaCap;
        // Localizăm nodul după care urmează să fie introdus
noul nod
        while (curent->next!=NULL && curent->next->data<
newNod->data)
        curent = curent->next;
        // Creăm legăturile corespunzătoare
        newNod->next = curent->next;
        // Dacă noul nod nu este inserat la sfârșitul listei
dublu înlănțuită
        if (curent->next != NULL)
           newNod->next->prev = newNod;
           curent->next = newNod; newNod->prev = curent;
    }
}
// Funcția pentru a sorta o listă dublu înlănțuită folosind
sortarea prin inserare
void insertionSort(struct Nod** referintaCap){
    //Inițializăm „sortat” - o listă sortată dublu înlănțuită
    struct Nod* sorted = NULL;
    // Parcurgem lista dublă și introducem fiecare nod în „sor-
tat”
    struct Nod* curent = *referintaCap;
    while (curent != NULL) {
        // Stocăm next pentru următoarea iterație
        struct Nod* next = curent->next;
        // Eliminarea tuturor legăturilor astfel încât să cream
un nou nod „curent” pentru inserare
        curent->prev = curent->next = NULL;
        // Inserăm „curent” în 'sortat'
        InsertionSort(&sorted, curent);
        curent = next; // Actualizăm „curent”
    }
    // Actualizăm referința pentru capul listei, pentru a in-
dica lista sortată dublu înlănțuită
    *referintaCap = sorted;
}
// Funcția de afișare a listei dublu înlănțuită
void afisareLista(struct Nod* capul){
    while (capul != NULL) {
        cout << capul->data << " "; capul = capul->next;
```

```
    }
}
// Funcția pentru a insera un nod la începutul listei duble
void inserare(struct Nod** head_ref, int new_data){
    // Alocăm nodul
    struct Nod* new_nod = (struct Nod*)malloc(sizeof(struct
Nod));
    new_nod->data = new_data; // Introducem datele
    // Facem următorul nou nod ca cap și anterior ca NUL
    new_nod->next = (*head_ref); new_nod->prev = NULL;
    // Schimbăm prev cu nodul principal în noul nod
    if ((*head_ref) != NULL) (*head_ref)->prev = new_nod;
    // Mutăm capul pentru a indica noul nod
    (*head_ref) = new_nod;
}
// Programul principal
int main(){
    // începem cu lista dublu înlănțuită goală
    struct Nod* capul = NULL;
    // introducem datele listei dublu înlănțuite
    inserare(&capul, 9); inserare(&capul, 3);
    inserare(&capul, 5); inserare(&capul, 10);
    inserare(&capul, 12); inserare(&capul, 8);
    cout << "\nLista dubla inainte de sortare:"<<endl;
    afisareLista(capul); insertionSort(&capul);
    cout << "\nLista dubla dupa de sortare:"<<endl;
    afisareLista(capul); return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Lista dubla inainte de sortare:
8 12 10 5 3 9
Lista dubla dupa de sortare:
3 5 8 9 10 12
```

```
Lista dubla inainte de sortare:
18 24 22 15 14 19
Lista dubla dupa de sortare:
14 15 18 19 22 24
```

3.5 Modele de probleme propuse

1. Elaborați subprograme în C++ pentru următoarele probleme simple.

- ***a.*** *Să se scrie o funcție care primește ca parametru adresa primului nod al listei și sterge primul și ultimul nod din listă.*
- ***b.*** *Să se scrie o funcție care primește ca parametru adresa primului nod al listei și interschimbă primul nod cu cel de-al doilea, al treilea - cu cel de-al patrulea ș.a.m.d.*
- ***c.*** *Să se scrie o funcție care primește ca parametru adresa ultimului nod al listei și mută al treilea nod în fața primului.*
- ***d.*** *Să se scrie o funcție care primește ca parametru adresa primului nod al listei și sterge primul număr par, al 3-lea număr par, al 5-lea număr par, al 7-lea număr par ș.a.m.d.*

2. Elaborați un program în C++ pentru următoarea problemă cu fișiere.

- ***a.*** *Să se creeze o listă liniară dublu înlănțuită cu noduri care să conțină în câmpul info număr întreg de maxim 4 cifre. Să se șteargă toate nodurile din listă cu excepția primului și a ultimului nod cu condiția că media lor este un întreg.*
- ***b.*** *Să se creeze o listă liniară dublu înlănțuită cu noduri care să conțină câmpurile prec și urm, care să conțină informații de legatură spre nodul anterior, respectiv următor din listă. Să se afișeze lista în ambele sensuri.*

3. Elaborați un program în C++ pentru următoarele probleme.

3.1 *Se consideră o listă dublu înlănțuită cu date despre elevi, cu structura: cod matricol, nume, prenume și media la BAC. Să se scrie un program în C++ pentru crearea, ordonarea după medie și afișarea listei elevilor folosindu-se câte o funcție adecvată pentru fiecare operație (creare, ordonare și listare).*

3.2 *Se consideră o listă dublu înlănțuită cu date despre fructe, cu structura: cod produs, denumire, țara de import și prețul unui kg. Să se scrie programul în C++ pentru crearea, ordonarea după preț și afișarea listei fructelor folosindu-se câte o funcție adecvată pentru fiecare operație (creare, ordonare și listare).*

3.3 *Se consideră o listă dublu înlănțuită cu date despre produse lactate, cu structura: cod produs, denumire, țara de import, prețul unui kg, % de grăsime, cantitatea de proteine la 100g. Să se scrie programul în C++ pentru crearea, ordonarea după cantitatea de proteine și afișarea listei produselor lactate folosindu-se câte o funcție adecvată pentru fiecare operație (creare, ordonare, listare și ștergere produs din listă).*

3.4 *Se consideră o listă dublu înlănțuită cu date despre 100 elevi, cu structura: cod matricol, nume și media din clasa 9-a. Să se scrie un program în C++ pentru crearea, actualizare (adăugări și ștergeri) și afișarea listei elevilor promovați:*

- *la bursă (media mai mare sau egală cu 8);*
- *la contract în fișier extern 1 (6 < media < 8);*
- *la școala profesională în fișier extern 2 (media < 6).*

3.5 *Se consideră o listă dublu înlănțuită cu date despre 100 antreprenori, cu structura: cod firmă, nume și venitul anual. Să se scrie un program în C++ pentru crearea, actualizare (adăugări și ștergeri) și afișarea listei antreprenorilor promovați:*

- *cu venit mare (venit mai mare decât un milion);*
- *cu venit mediu în fișier extern 1 (venit < 500.000);*
- *cu datorii în fișier extern 2 (venit < 0).*

3.6 *Se consideră o listă dublu înlănțuită cu date despre 100 sportivi, cu structura: cod sport, nume, prenume, denumirea sportului, % masei musculare și nr. de calorii medii arse pe zi la sală. Să se scrie un program în C++ pentru crearea, actualizare (adăugări și ștergeri) și afișarea listei sportivilor promovați:*

- *cu masă musculară mare (% masei musculare >50%);*
- *cu masă musculară mică în fișier extern cu denumirea sportivi.out.*

4. Probleme suplimentare pentru LLDÎ

În exercițiile din acest compartiment se va folosi o listă avînd ca informație utilă în nod un număr real:

```
typedef struct TNOD{
        float x;
        struct TNOD *next, *pred;
};
```

1. *Să se scrie programul pentru crearea unei liste dublu înlănțuite cu preluarea datelor de la tastatură. Sfîrșitul introducerii datelor este marcat standard. După creare, se va afișa conținutul listei, apoi se va elibera memoria ocupată.*
2. *Să se scrie funcția pentru inserarea unui nod la începutul unei liste dublu înlănțuite. Funcția are ca parametri capul listei și valoarea care trebuie inserată. Prin numele funcției se întoarce noul cap al listei.*
3. *Să se scrie funcția pentru inserarea unui nod într-o listă dublu înlănțuită, după un nod identificat prin valoarea unui câmp. Funcția are ca parametri capul listei, valoarea care se inserează și valoarea după care se inserează. Dacă valoarea căutată nu este găsită, se face inserare la sfârșitul listei.*
4. *Să se scrie funcția pentru inserarea unui nod într-o listă dublu înlănțuită, înaintea unui nod identificat prin valoarea unui cîmp. Funcția are ca parametri capul listei, valoarea care trebuie inserată și valoarea înaintea căreia se inserează. Dacă valoarea căutată nu este găsită, se face inserare la începutul listei.*
5. *Să se scrie funcția pentru căutarea unui nod identificat prin valoarea unui cîmp într-o listă dublu înlănțuită. Funcția are ca parametri capul listei în care se caută și valoarea căutată. Prin numele funcției se întoarce adresa nodului care conține informația căutată sau NULL dacă informația nu a fost găsită în listă. Dacă sunt mai multe noduri cu aceeași valoare se întoarce adresa ultimului. Funcția este identică pentru liste simplu și dublu înlănțuite.*
6. *Să se scrie funcția pentru adăugarea unui nod la sfârșitul unei liste dublu înlănțuite. Funcția are ca parametri capul listei și valoarea care trebuie inserată. Funcția întoarce, prin numele ei, noul cap al listei.*

7. *Să se scrie funcţia pentru ştergerea primului nod al unei liste dublu înlănţuite. Funcţia are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei.*
8. *Să se scrie funcţia pentru ştergerea unui nod identificat prin valoarea unui câmp dintr-o listă dublu înlănţuită. Funcţia are ca parametri capul listei şi valoarea de identificare a nodului care trebuie şters. Prin numele funcţiei se întoarce noul cap al listei.*
9. *Să se scrie funcţia pentru ştergerea ultimului nod dintr-o listă dublu înlănţuită. Funcţia are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei.*
10. *Să se scrie funcţia pentru ştergerea unei liste dublu înlănţuite. Funcţia are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei (valoarea NULL).*
11. *Să se scrie funcţia pentru ştergerea nodului aflat după un nod identificat prin valoarea unui câmp. Funcţia are ca parametri capul listei şi valoarea căutată. Prin numele funcţiei se întoarce parametrul de eroare, cu semnificaţia: 0 – s-a efectuat ştergerea, 1 – nodul căutat nu a fost găsit.*
12. *Să se scrie funcţia pentru ştergerea nodului aflat înaintea nodului identificat prin valoarea unui câmp. Funcţia are ca parametri adresa capului listei şi valoarea căutată. Prin numele funcţiei se întoarce parametrul de eroare, cu semnificaţia: 0 – s-a efectuat ştergerea, 1 – valoarea căutată nu a fost găsită.*
13. *Să se scrie funcţia pentru ştergerea nodului aflat după un nod identificat prin valoarea unui câmp. Funcţia are ca parametri capul listei şi valoarea nodului căutat (după care se şterge). Prin numele funcţiei se întoarce valoarea 0, dacă s-a făcut ştergerea, sau 1, dacă nodul căutat nu a fost găsit.*
14. *Să se scrie funcţia pentru ştergerea nodului aflat înaintea nodului identificat prin valoarea unui câmp. Funţia are ca parametri capul listei, valoarea din nodul căutat (înaintea căruia se face ştergerea) şi adresa unde se înscrie parametrul de eroare (0 dacă se face ştergerea, 1 dacă nodul căutat este primul, 2 dacă nodul căutat nu a fost găsit). Funcţia întoarce noul cap al listei.*

3.6 Portofoliul elevului pentru LLDÎ

Se consideră o listă liniară dublu înlănțuită care memorează valori întregi. Elaborați subprograme pentru următoarele operațiuni:

a. Introduceți un număr nou în listă;
b. Afișați elementele pare și impare ale listei;
c. Afișați elementele pare pozitive și pare negative ale listei;
d. Afișați elementele impare pozitive și impare negative ale listei;
e. Afișați elementele în ordine crescătoare ale listei;
f. Afișați elementele în ordine descrescătoare ale listei;
g. Să se șteargă primul nod (ultimul nod)care conține valoarea x.
h. Să se șteargă nodurile care conțin pătrate perfecte.
i. Să se șteargă nodurile care conțin numere Fibonacci.

Fiecare elev din subgrupă va realiza o bibliotecă ce va include operațiile de mai sus, cu condiția că elementele inițiale ale listei sunt citite din fișier, fiecare elev va avea o listă individuală conform numărului de ordine.

Nr.	Secvența de numere	Nr.	Secvența de numere
1	50 31 30 35 82 85 16 18 58 61	**11**	21 37 56 32 44 71 99 87 72 90
2	79 59 45 14 57 23 73 83 51 62	**12**	19 12 49 18 37 71 39 68 42 98
3	77 81 66 84 95 43 38 65 15 76	**13**	75 46 97 40 57 70 54 73 14 59
4	26 64 39 13 17 10 40 55 29 48	**14**	56 31 44 27 96 80 81 86 58 65
5	47 74 27 91 75 88 33 25 93 42	**15**	53 21 50 67 83 69 93 32 43 17
6	67 22 69 96 46 97 28 12 41 36	**16**	41 95 89 77 29 33 87 51 99 72
7	94 63 92 34 53 78 11 52 19 86	**17**	61 76 45 24 79 85 52 92 15 36
8	70 24 89 80 60 49 68 20 98 54	**18**	82 64 74 38 23 16 63 30 10 62
9	90 60 11 78 35 84 22 88 20 34	**19**	62 12 38 79 43 90 17 23 80 92
10	55 48 28 26 66 25 91 94 13 47	**20**	30 42 50 49 41 11 71 58 83 44

3.7 Model de test grilă pentru LLDÎ

Nr.	Conţinutul	Punctaj
1	*O componentă a unei liste dublu înlănțuite se declară ca o dată structurată de tip înregistrare, formată din trei câmpuri: informația propriu-zisă, adresa la care e memorată următoarea componentă și adresa la care e memorată precedenta componentă.* ◯ adevărat ◯ fals	*2p*
2	*Pentru inserarea la începutul listei trebuie doar alocat elementul, legat de ultimul element din listă și repoziționarea capului listei.* ◯ adevărat ◯ fals	*2p*
3	*Fie că avem o funcție care primește ca parametri doi pointeri la începuturile a două liste dublu înlănțuite sortate ascendent și întoarce o listă dublu înlănțuită sortată ascendent ce conține reuniunea dintre cele două liste. De exemplu*: pentru listele *A: 1⟺2⟺5* și *B: 2⟺3⟺7*, ce va rezulta? ◯ 1⇔2⇔2⇔3⇔5⇔7 ◯ 1⇔2⇔3⇔5⇔7 ◯ 7⇔5⇔3⇔2⇔2⇔1 ◯ 7⇔5⇔3⇔2⇔1	*4p*
4	*Fie că avem o funcție care primește ca parametri doi pointeri la începuturile a două liste dublu înlănțuite sortate ascendent și întoarce o listă dublu înlănțuită sortată ascendent ce conține diferența, B-A, dintre cele două liste. De exemplu*: pentru listele *A: 1⟺2⟺5* și *B: 2⟺3⟺7*, ce va rezulta? ◯ 3⇔7 ◯ 2 ◯ 5⇔1 ◯ 7⇔3	*4p*
5	*Fie că avem o funcție care primește ca parametri doi pointeri la începuturile a două liste dublu înlănțuite sortate și întoarce o listă dublu înlănțuită sortată ascendent ce conține fracțiile reductibile formate din elementele celor două liste. De exemplu*: pentru listele *A: 1⟺2⟺5* și *B: 2⟺8⟺10*, ce va rezulta? ◯ 1/2⇔2/8⇔5/10 ◯ 2/8⇔5/10 ◯ 1/2⇔1/4⇔1/2 ◯ 1/4⇔1/2	*4p*

6	*Fiind date două liste dublu înlănțuite, A şi B, construiţi o nouă listă dublu înlănțuită C, pentru care fiecare nod i din C reţine suma dintre valoarea nodului i din A şi valoarea nodului i din B. Dacă una dintre listele primite este mai lungă decât cealaltă, se consideră că nodurile asociate lipsă din cealaltă listă conţin valoarea 0, adică se păstrează valorile din lista mai lungă. De exemplu: pentru lista A: 3⟺7⟺29⟺4 şi lista B: 2⟺4⟺3*, ce va rezulta? ◯ C: 5⇔11⇔32⇔4 ◯ C: 0⇔5⇔11⇔32 ◯ C: 1⇔3⇔26⇔4 ◯ C: 5⇔11⇔32⇔0	**4p**
7	*Un element poate fi căutat în lista liniară dublu înlănțuită în următoarele cazuri:* ☐ după poziție ☐ dacă lista este vidă ☐ după valoare ☐ alt răspuns	**2p**
8	*Un element poate fi inserat în lista liniară dublu înlănțuită în următoarele cazuri:* ☐ la început ☐ la sfârșit ☐ după un element ☐ alt răspuns	**3p**
9	*Un element poate fi șters din lista liniară dublu înlănțuită în următoarele moduri:* ☐ din interior (diferit de capul listei) ☐ după o anumită valoare ☐ de pe o anumită poziție ☐ răspunsul corect lipsește	**3p**

Barem de notare propus

Nota	10	9	8	7	6	5	4	3	2
%	100-95	94-88	87-78	77-63	62-48	47-33	32-21	20-10	9-5
Punctaj	28-27	26-25	24-22	21-18	19-13	12-9	8-6	5-3	2-1

4. LISTE CIRCULARE

4.1 Introducere

Lista circulară este o listă legată, în care toate nodurile sunt conectate pentru a forma un cerc. La final nu există NULL. O listă circulară poate fi o listă simplu înlănțuită sau o listă dublu înlănțuită:

- *O **listă circulară simplu înlănțuită** este o listă liniară simplu înlănțuită modificată astfel încât ultimul element pointează (tinde) spre primul element din listă.*

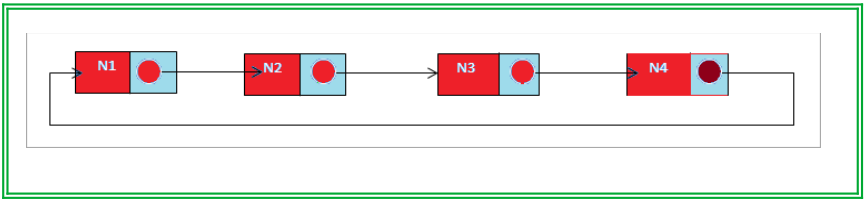


Figura 3.1 – Reprezentarea grafică a unei liste circulare simplu înlănțuite

- *O **listă circulară dublu înlănțuită** este o listă liniară dublu înlănțuită modificată astfel încât ultimele elemente pointează respectiv spre primele elemente din listă.*

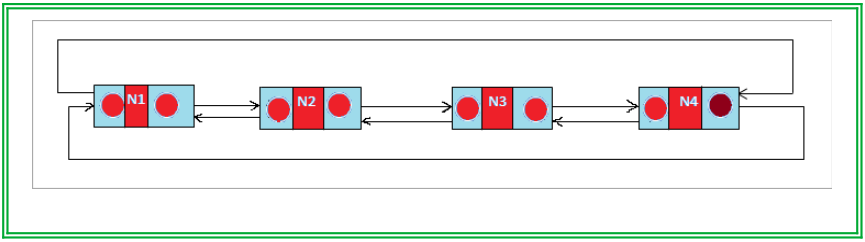


Figura 3.2 – Reprezentarea grafică a unei liste circulare dublu înlănțuite

Avantajele listelor circulare înlănțuite:

- *Orice nod poate fi un punct de plecare. Putem traversa întreaga listă pornind de la orice punct. Trebuie doar să ne oprim când primul nod vizitat este vizitat din nou.*
- *Util pentru implementarea cozii. Spre deosebire de această implementare, nu este necesar să menținem două indicatoare: pentru față și spate dacă folosim o listă circulară înlănțuită. Putem menține un pointer către ultimul nod introdus și frontal poate fi obținut întotdeauna ca următorul.*
- *Listele circulare sunt utile în aplicații pentru a merge în mod repetat în jurul listei. De exemplu, când mai multe aplicații rulează pe un computer, este obișnuit ca sistemul de operare să pună aplicațiile rulante pe o listă și apoi să le parcurgă, oferindu-le fiecăreia o porție de timp pentru a le executa, apoi făcându-le să aștepte, în timp ce procesorul este dat unei alte aplicații. Este convenabil ca sistemul de operare să utilizeze o listă circulară, astfel încât, atunci când ajunge la sfârșitul listei, să se poată deplasa în fața listei.*
- *Listele circulare dublu înlănțuite sunt utilizate pentru implementarea structurilor avansate de date, cum ar fi Fibonacci Heap.*

Unele aplicații ale listelor circulare:

- *Jocurile care implică mai mulți jucători pot fi, de asemenea, reprezentate folosind o listă circulară în care fiecare jucător este un nod căruia i se oferă o șansă de a juca.*
- *Putem utiliza o listă circulară pentru a reprezenta o coadă circulară. Procedând astfel, putem elimina cei doi indicatori față și spate care sunt folosiți pentru coadă. În schimb, putem folosi un singur indicator etc.*

4.2 Structura unei liste circulare înlănțuite

O listă circulară este o colecție de noduri în care nodurile sunt conectate între ele pentru a forma un cerc. Aceasta înseamnă că în loc de a seta următorul indicator al ultimului nod la null, acesta este legat de primul nod.

O asemenea listă este utilă în reprezentarea structurilor sau activităților care trebuie repetate iar și iar după un interval de timp specific, cum ar fi programele dintr-un mediu multiprogramare. De asemenea, este benefic pentru proiectarea unei cozi circulare.

Listele circulare acceptă diferite operațiuni precum inserarea, ștergerea și traversările. Putem declara un nod dintr-o listă circulară ca orice alt nod, după cum se arată mai jos:

```
// Datele asociate unui element (nod) dintr-o listă
typedef int Date;
```

În cazul în care se dorește memorarea unui alt tip de date, trebuie schimbată doar declarația aliasului Date. Pentru memorarea listei se folosește o structură autoreferită. Acestă structură va avea forma:

```
struct Element{
   Date valoare; // datele efective memorate
   struct Element* next; // legatura către nodul următor
};
```

*Pentru a implementa o listă circulară, menținem un indicator extern **last** care indică ultimul nod din lista circulară. De aici **last-> next** va indica primul nod din listă.*

Procedând astfel, ne asigurăm că atunci când introducem un nou nod la începutul sau la sfârșitul listei, nu trebuie să traversăm întreaga listă. Acest lucru nu ar fi fost posibil dacă am fi indicat indicatorul extern către primul nod.

4.3 Operații principale efectuate

4.3.1 Parcurgerea listei circulare

Parcurgerea este o tehnică de vizitare a fiecărui nod. În listele liniare conectate, precum listele simplu și listele dublu înlănțuite, parcurgerea este ușoară pe măsură ce vizităm fiecare nod și ne oprim când este întâlnit ***NULL****. Totuși, acest lucru nu este posibil într-o listă circulară legată. Pentru a parcurge o listă circulară legată, începem de la nodul* ***ultim,*** *care reprezintă primul nod al listei. Finisăm parcurgerea listei când ajungem la nodul inițial.*

Acum prezentăm o implementare a operațiunilor de listă circulară în C++. Am implementat aici operațiuni de inserare, ștergere și traversare (parcurgere). Pentru o înțelegere clară, am reprezentat o listă circulară astfel: ***10→20→30→40→50→10.*** *Astfel la ultimul nod 50, atașăm din nou nodul 10 care este primul nod, indicându-l astfel ca o listă circulară.*

```
// Parcurgerea listei circulare
void traverseList(struct Node *last) {
     struct Node *p;
     // Dacă lista este goală, revenim.
     if (last == NULL) {
         cout << "Lista circulară este goală." << endl;
         return;
     }
     // Indicăm primul nod din listă
     p = last -> next;
     // Parcurgem lista circulară începând cu primul nod
     // până când ultimul nod este vizitat din nou.
     do {
        cout << p -> data << "==>"; p = p -> next;
     } while(p != last->next);
     if(p == last->next) cout<<p->data; cout<<"\n\n";
}
```

4.3.2 Inserarea (adăugarea) unui element în listă

Putem introduce un nod într-o listă circulară fie ca primul nod (lista goală), la început, la sfârșit, sau între celelalte noduri.

Inserarea într-o listă goală

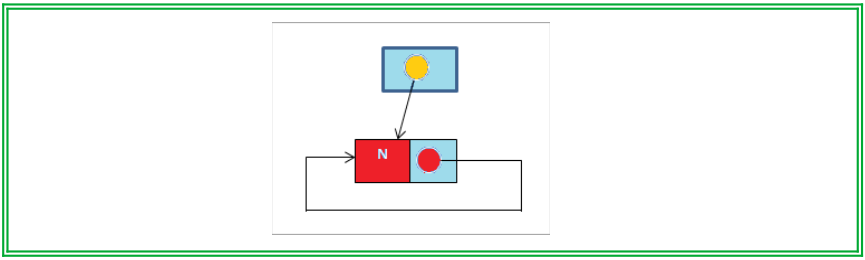


Figura 3.3 – Reprezentarea grafică a inserării unui nod în lista circulară

Când nu există noduri în lista circulară și lista este goală, ultimul indicator este nul, atunci introducem un nou nod N, indicând ultimul indicator către nodul N, așa cum se arată în imaginea alăturată. Următorul indicator al lui N va indica nodul N în sine, întrucât există un singur nod. Astfel N devine primul nod și ultimul nod din listă.

```
//Inserarea unui nou nod într-o listă goală
struct Node *insertInEmpty(struct Node *last, int new_data){
   // Dacă ultimul nod nu este nul,
   // atunci lista nu este goală, revenim
   if (last != NULL) return last;
   // Alocăm memorie pentru nod
   struct Node *temp = new Node;
   // Setăm date pentru nodul nou
   temp -> data = new_data;
   last = temp;
   // Creăm legătura
   last->next = last;
   // Returnăm ultimul
   return last;
}
```

Inserarea la începutul listei

Atunci când adăugăm un nod la începutul listei, următorul indicator al ultimului nod indică noul nod N, făcându-l astfel un prim nod. Pentru a insera un nou nod la începutul listei, urmăm acești pași:

- *N->next = last->next;*
- *last->next = N.*

O reprezentare grafică a acestei operațiuni este mai jos precum și secvența de cod C++:

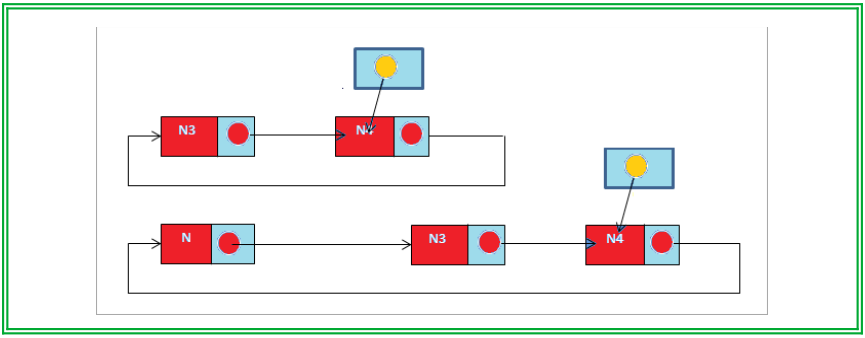


Figura 3.4 – Reprezentarea grafică a inserării unui nod la începutul listei circulare

```
// Inserarea unui nod nou la începutul listei
struct Node *insertAtBegin(struct Node *last, int new_data){
   // Dacă lista este goală, atunci adăugăm nodul apelând
   // la insertInEmpty
   if (last == NULL)
   return insertInEmpty(last, new_data);
   // În caz contrar creăm un nou nod
   struct Node *temp = new Node;
   // Setăm date pentru nodul nou
   temp -> data = new_data;
   temp -> next = last -> next;
   last -> next = temp; return last;
}
```

Inserarea la sfârșitul listei

Pentru a insera un nou nod la sfârșitul listei, urmăm acești pași:

- *N-> next = last ->next;*
- *last ->next = N; last = N.*

O reprezentare grafică a acestei operațiuni este mai jos precum și secvența de cod C++:

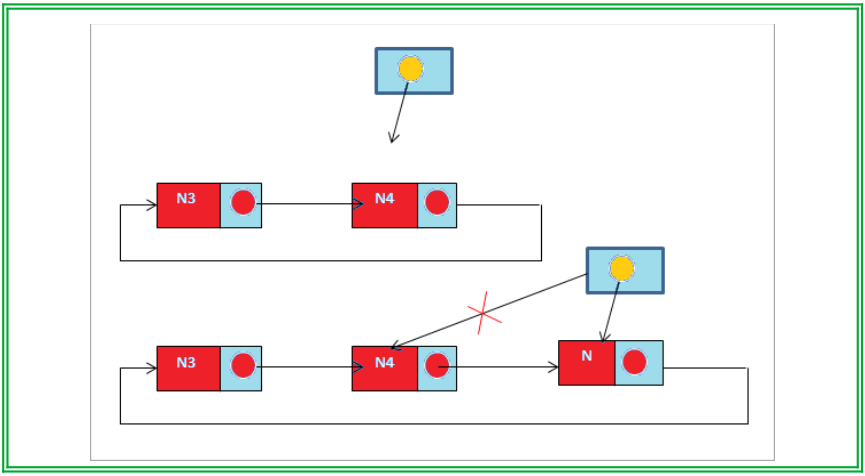


Figura 3.5 – Reprezentarea grafică a inserării unui nod la sfârșitul listei circulare

```
// Inserarea unui nod nou la sfârșitul listei
struct Node *insertAtEnd(struct Node *last, int new_data){
   // Dacă lista este goală, atunci adăugăm nodul apelând
   // la insertInEmpty
   if (last == NULL)
   return insertInEmpty(last, new_data);
   // În caz contrar creăm un nou nod
   struct Node *temp = new Node;
   // Setăm date pentru nodul nou
   temp -> data = new_data; temp -> next = last -> next;
   last -> next = temp; last = temp; return last;
}
```

Inserare în interiorul listei

Să presupunem că trebuie să introducem un nou nod N între nodurile N3 și N4. Mai întâi trebuie să traversăm toată lista. În timp ce parcurgem lista trebuie să localizăm nodul, după care urmează să fie inserat noul nod, în acest caz, N3. După localizarea nodului, efectuăm următorii pași:

- *N -> next = N3 -> next;*
- *N3 -> next = N.*

Aceasta secvență de cod permite introducerea unui nou nod N după nodul N3, pentru mai multe detalii analizați reprezentarea grafică, precum și secvența de cod C++:

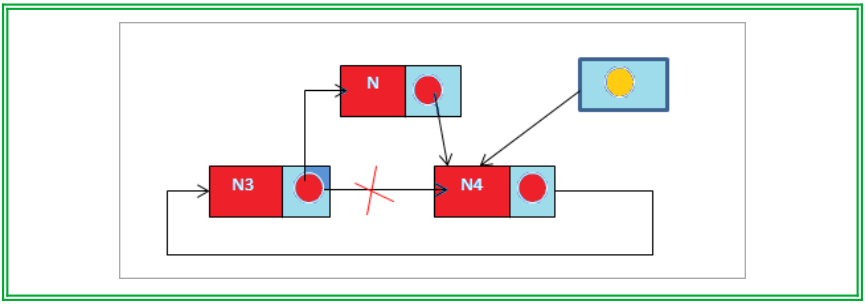


Figura 3.6 – Reprezentarea grafică a inserării unui nod în interiorul listei circulare

```
// Inserarea în interiorul listei
struct Node *insertAfter(struct Node *last, int new_data, int
after_item) {
   // Returnează null dacă lista este goală
   if (last == NULL) return NULL;
   struct Node *temp, *p; p = last -> next;
   do {
      if (p ->data == after_item){
         temp = new Node; temp -> data = new_data;
         temp -> next = p -> next; p -> next = temp;
         if (p == last) last = temp; return last;
       } p = p -> next;
    } while(p != last -> next);
    cout << "Nodul cu data "<<after_item << " nu este prezent
```

```
in lista." << endl;
    return last;
}
```

4.3.3 Ștergerea unui element din listă

*Operația de ștergere a listei circulare presupune localizarea nodului care urmează să fie șters și apoi eliberarea memoriei sale. Pentru aceasta menținem două indicatoare suplimentare **curr** și **prev** (nod anterior), apoi traversăm lista pentru a localiza nodul.*

*Nodul dat care va fi șters poate fi primul nod, ultimul nod sau nodul dintre ele. În funcție de locație, setăm indicatoarele **curr** și **prev** și apoi ștergem nodul **curr** (nod actual).*

O reprezentare grafică a acestei operațiuni este mai jos precum și secvența de cod C++:

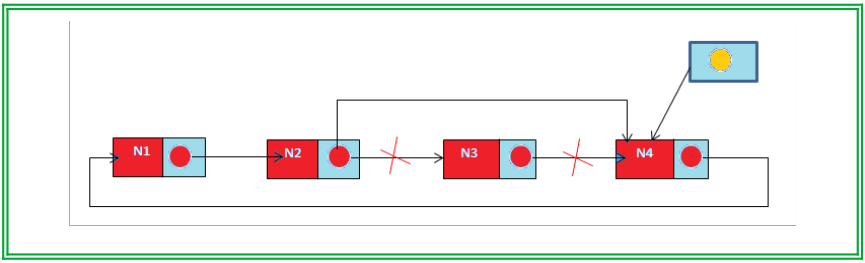


Figura 3.7 – Reprezentarea grafică a ștergerii unui nod din lista circulară

```
// Ștergerea nodului din listă
void deleteNode(Node** head, int key){
   // Dacă lista este goală, revenim
   if (*head == NULL) return;
   // Dacă lista conține un singur nod, ștergem acel nod
   // Lista este goală
   if((*head)->data==key && (*head)->next==*head) {
      free(*head); *head=NULL;
    }
    Node *last=*head,*d;
    // Dacă cheia este capul
```

```
    if ((*head)->data==key) {
       while(last->next!=*head)
       // Găsim ultimul nod al listei
       last=last->next;
       // Indicăm ultimul nod către următorul cap sau
       // al doilea nod al listei
       last->next=(*head)->next; free(*head);
       *head=last->next;
    }
    // Sfârșitul listei este atins sau
    // nodul care trebuie șters nu există în listă
    while (last->next!=*head&&last->next->data!=key) {
          last=last->next;
    }
    // Se găsește nodul care trebuie șters
    // Eliberăm astfel memoria și afișăm lista
    if (last->next->data==key) {
       d=last->next; last->next=d->next;
            cout<<"Nodul  cu  data  "<<key<<"  este  sters  din
lista!"<<endl;
       free(d); cout<<endl;
       cout<<"Lista circulara după ștergere "<<key<<" este după
cum urmează:"<<endl;
        traverseList(last);
      }
    else
     cout<<"Nodul  cu  data  "<<  key  <<  "  nu  se  gaseste  in
lista!"<<endl;
}
```

Remarcă:

- *Dacă vă mai aduceți aminte cum se realizează un exemplu de fișier antet, creați un asemenea fișier pentru structura dinamică respectivă, incluzând operațiile studiate în acest compartiment.*
- *Pentru crearea unui asemenea fișier, este necesar să analizați pașii prezentați în exemplele din anexa 1.*
- *De asemenea la dorință se poate de îmbunătățit acest fișier antet cu alte subprograme pe care le veți aplica la rezolvarea problemelor.*

4.4 Modele de probleme rezolvate

Problema 1 – Operații fundamentale ale LCSÎ

Vom prezenta o implementare a operațiunilor unei liste circulare în C++, aplicând operațiuni de inserare, ștergere și traversare (parcurgere) a listei circulare simplu înlănțuite.

Prezentarea soluției problemei în limbajul C++

```
#include<iostream>
using namespace std;
struct Node{
    int data;
    struct Node *next;
};
// Inserarea unui nod nou intr-o lista goala
struct Node *insertInEmpty(struct Node *last, int new_data){
    if (last != NULL) return last;
    struct Node *temp = new Node;
    temp -> data = new_data; last = temp;
    last->next = last; return last;
}
// Inserarea unui nod nou la inceput de lista
struct Node *insertAtBegin(struct Node *last, int new_data){
    if (last == NULL)
    return insertInEmpty(last, new_data);
    struct Node *temp = new Node;
    temp -> data = new_data; temp -> next = last -> next;
    last -> next = temp; return last;
}
// Inserarea unui nod nou la sfarsit de lista
struct Node *insertAtEnd(struct Node *last, int new_data){
    if (last == NULL)
    return insertInEmpty(last, new_data);
    struct Node *temp = new Node;
    temp -> data = new_data; temp -> next = last -> next;
    last -> next = temp; last = temp; return last;
```

```
}
// Inserarea unui nod nou in interiorul unei liste
struct Node *insertAfter(struct Node *last, int new_data, int
after_item){
    if (last == NULL) return NULL;
    struct Node *temp, *p; p = last -> next;
    do{
        if (p ->data == after_item){
            temp = new Node; temp -> data = new_data;
            temp -> next = p -> next; p -> next = temp;
            if (p == last) last = temp; return last;
        }
    p = p -> next;
    } while(p != last -> next);
      cout << "Nodul cu valoarea "<<after_item << " nu este
prezent in lista." << endl;
    return last;
}
// Parcurgerea listei circulare
void traverseList(struct Node *last) {
    struct Node *p;
    if (last == NULL) {
        cout << "Lista circulara este goala!" << endl;
        return;
    }
    p = last -> next;
    do{
        cout << p -> data << " ==> "; p = p -> next;
    } while(p != last->next);
    if(p == last->next) cout<<p->data; cout<<"\n\n";
}
// Stergerea unui nod din lista circulara
void deleteNode(Node** head, int key){
    if (*head == NULL) return;
    if((*head)->data==key && (*head)->next==*head) {
      free(*head); *head=NULL;
    }
    Node *last=*head,*d;
    if((*head)->data==key) {
        while(last->next!=*head)
        last=last->next; last->next=(*head)->next;
        free(*head); *head=last->next;
```

```
    }
    while(last->next!=*head&&last->next->data!=key) {
        last=last->next;
    }
    if(last->next->data==key){
        d=last->next; last->next=d->next;
          cout<<"Nodul cu valoarea "<<key<<" a fost sters din
lista!"<<endl;
        free(d); cout<<endl;
         cout<<"Lista circulara dupa stergerea nodului cu val-
oarea "<<key<<" este urmatoarea:"<<endl;
        traverseList(last);
    }
    else cout<<"Nodul cu valoarea "<< key << " nu a fost gasit
in lista."<<endl;
}
// Programul principal
int main(){
    struct Node *last = NULL;
    last = insertInEmpty(last, 30);
    last = insertAtBegin(last, 20);
    last = insertAtBegin(last, 10);
    last = insertAtEnd(last, 40);
    last = insertAtEnd(last, 60);
    last = insertAfter(last, 50,40 );
    cout<<"Lista circulara creata este urmatoarea: "<<endl;
    traverseList(last); deleteNode(&last,10);
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Lista circulara creata este urmatoarea:
10 ==> 20 ==> 30 ==> 40 ==> 50 ==> 60 ==> 10

Nodul cu valoarea 10 a fost sters din lista!

Lista circulara dupa stergerea nodului cu valoarea 10 este ur-
matoarea:
20 ==> 30 ==> 40 ==> 50 ==> 60 ==> 20
```

Problema 2 – Conversia unei LLSÎ în LCSÎ

Fie avem o listă liniară simplu înlănțuită. Să se elaboreze un program care să convertească lista simplă în listă circulară.

De exemplu, ni s-a dat o listă simplă cu 6 noduri și dorim să convertim această listă simplă în listă circulară.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
/* Nodul listei simple */
struct Nod {
    int data;
    struct Nod* next;
};
// Funcția de convergere a unei LLSÎ în LCSÎ.
struct Nod* ListaCirculara(struct Nod* cap){
    // Declarăm un nod start și atribuim nodul principal nodu-
lui de pornire.
    struct Nod* start = cap;
    // Verificăm dacă în timp ce cap-> next nu este egal cu
NULL, apoi cap indică următorul nod.
    while (cap->next != NULL)
        cap = cap->next;
    // Dacă cap->next indică spre NULL, atunci începem să
atribuim cap->next, nodul următor.
    cap->next = start; return start;
}
// Funcția de adăugare a unui nou nod în listă
void miscare(struct Nod** cap, int data){
    // Alocăm memorie dinamică pentru noul nod, NodNou.
    struct Nod* NodNou = (struct Nod*)malloc(sizeof(struct
Nod));
    // Alocăm datele în noul nod, NodNou.
    NodNou->data = data;
    // NodNou->next alocăm adresa nodului cap
    NodNou->next = (*cap);
    // NodNou devine nod cap.
```

```
    (*cap) = NodNou;
}
// Funcția care afișează elementele listei circulare
void afisareLista(struct Nod* nod){
    struct Nod* start = nod;
    while (nod->next != start) {
        cout<<nod->data<<" ";
        nod = nod->next;
    }
    // Afișăm ultimul nod al listei circulare.
    cout<<nod->data<<" ";
}
// Programul principal pentru testarea funcțiilor elaborate
int main(){
    // Începem cu o listă goală
    struct Nod* cap = NULL;
    cout<<"Lista este: \n\t";
    // Utilizăm funcția miscare() pentru a construi LLSÎ: 19-
>17->22->13->14->15
    miscare(&cap, 15); miscare(&cap, 14); miscare(&cap, 13);
    miscare(&cap, 22); miscare(&cap, 17); miscare(&cap, 19);
    // Apelăm funcția ListaCirculara care convertește LLSÎ în
LCSÎ.
    ListaCirculara(cap);
    cout<<"\nAfisarea elementelor listei circulare: \n\t";
afisareLista(cap); return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Lista este:

Afisarea elementelor listei circulare:
        19 17 22 13 14 15
```

```
Lista este:

Afisarea elementelor listei circulare:
        15 14 13 22 17 19
```

Problema 3 – Modalități de inserare a elementelor în LCSÎ

Fie avem o listă circulară simplu înlănțuită. Să se elaboreze subprograme pentru a insera un element: într-o listă goală, la începutul listei, la sfârșitul listei și între noduri.

Prezentarea soluției problemei în limbajul C++

```
#include<iostream>
using namespace std;
struct Nod{
    int data;
    struct Nod *next;
};
// Funcția ce permite inserarea unui nod într-o listă goală
struct Nod *inserareaGoala(struct Nod *ultim, int data){
    if (ultim != NULL) return ultim;
    // Crearea unui nod dinamic.
    struct Nod *temp = (struct Nod*)malloc(sizeof(struct Nod));
    // Atribuirea datelor listei.
    temp -> data = data; ultim = temp;
    // Crearea legăturii.
    ultim -> next = ultim; return ultim;
}
// Funcția ce permite inserarea unui nod la începutul LCSÎ
struct Nod *inserareaInainte(struct Nod *ultim, int data){
    if (ultim == NULL) return inserareaGoala(ultim, data);
    struct Nod *temp = (struct Nod *)malloc(sizeof(struct
Nod));
    temp -> data = data; temp -> next = ultim -> next;
    ultim -> next = temp; return ultim;
}
// Funcția ce permite inserarea unui nod la sfârșitul LCSÎ
struct Nod *inserareaSfarsit(struct Nod *ultim, int data){
    if (ultim == NULL) return inserareaGoala(ultim, data);
    struct Nod *temp = (struct Nod *)malloc(sizeof(struct
Nod));
    temp -> data = data; temp -> next = ultim -> next;
    ultim -> next = temp; ultim = temp; return ultim;
}
```

```
// Funcția ce permite inserarea unui nod după un alt nod din
LCSÎ
struct Nod *inserareaDupa(struct Nod *ultim, int data, int
item){
    if (ultim == NULL) return NULL;
    struct Nod *temp, *p;
    p = ultim -> next;
    do {
        if (p ->data == item){
            temp = (struct Nod *)malloc(sizeof(struct Nod));
            temp -> data = data; temp -> next = p -> next;
            p -> next = temp;
            if (p == ultim) ultim = temp;
            return ultim;
        }
        p = p -> next;
    }  while(p != ultim -> next);
    cout << item << " nu este prezent in lista!" << endl;
    return ultim;
}
// Funcția ce permite afișarea unei LCSÎ
void afisareLista(struct Nod *ultim){
    struct Nod *p;
    // Verificăm dacă lista este sau nu goală
    if (ultim == NULL) {
        cout << "\tLista circulara este goala!" << endl;
        return;
    }
    // Fixăm primul nod al listei.
    p = ultim -> next; cout<<"\t";
    // Parcurgerea elementelor listei.
    do {
        cout << p -> data << " "; p = p -> next;
    }
    while(p != ultim->next);
}
// Programul principal
int main(){
    struct Nod *ultim = NULL;
    cout<<"Lista initiala: \n";afisareLista(ultim);
    cout<<"Lista dupa inserarea unui nod: \n";
    ultim = inserareaGoala(ultim, 6); afisareLista(ultim);
```

```
    cout<<"\nLista dupa inserarea a doua noduri inainte: \n";
    ultim = inserareaInainte(ultim, 4);
    ultim = inserareaInainte(ultim, 2); afisareLista(ultim);
    cout<<"\nLista dupa inserarea a doua noduri in urma: \n";
    ultim = inserareaSfarsit(ultim, 8);
    ultim = inserareaSfarsit(ultim, 12); afisareLista(ultim);
    cout<<"\nLista dupa inserarea unui nod dupa un alt nod din
lista: \n"; ultim = inserareaDupa(ultim, 10, 8);
    // 10 se plasează după 8 în LCSÎ
    afisareLista(ultim);
    cout<<"\nLista dupa inserarea unui nod dupa un alt nod din
lista: \n"; ultim = inserareaDupa(ultim, 3, 2);
    // 3 se plasează după 2 în LCSÎ
    afisareLista(ultim); return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Lista initiala:
        Lista circulara este goala!
Lista dupa inserarea unui nod:
        6
Lista dupa inserarea a doua noduri inainte:
        2 4 6
Lista dupa inserarea a doua noduri in urma:
        2 4 6 8 12
Lista dupa inserarea unui nod dupa un alt nod din lista:
        2 4 6 8 10 12
Lista dupa inserarea unui nod dupa un alt nod din lista:
        2 3 4 6 8 10 12
```

Problema 4 – Afișare min și max din LCSÎ

Fie avem o listă circulară simplu înlănțuită. Să se elaboreze subprograme pentru a afișa elementul maxim și cel minim din listă.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
/* Nodul listei simple */
struct Nod {
    int data;
    struct Nod* next;
};
// Funcția ce afișează nodul minim și maxim al listei circulare
void afisareMinMax(struct Nod** cap){
    // Verificăm dacă lista este goală
    if (*cap == NULL) {
        return;
    }
    struct Nod* curent;
    // Inițializăm capul spre indicatorul curent
    curent = *cap;
    // Inițializăm valoarea max-int la min și valoarea min-int
la max
    int min = INT_MAX, max = INT_MIN;
    // În timp ce ultimul nod nu este atins
    do {
        // Dacă datele din nodul curent sunt mai mici pentru
min, atunci le înlocuim
        if (curent->data < min) {
            min = curent->data;
        }
        // Dacă datele din nodul curent sunt mai mari pentru
max, atunci le înlocuim
        if (curent->data > max) {
            max = curent->data;
        }
        curent = curent->next;
    }while(curent != *cap);
    cout << "\n\tNodul minimum = " << min;
```

```
   cout << "\n\tNodul maximum = " << max;
}
// Funcția de a insera un nod la sfârșitul unei liste circulare
void InserareNod(struct Nod** cap, int data){
    struct Nod* curent = *cap;
    // Creăm un nou nod
    struct Nod* NouNod = new Nod;
    // Verificăm dacă nodul a fost creat sau nu
    if (!NouNod) {
        cout << "\nEroare de memorie!\n"; return;
    }
    // Introducem date în nodul nou creat
    NouNod->data = data;
    // Verificăm dacă lista circulară este goală. Dacă nu aveți
niciun nod, atunci creăm primul nod
    if (*cap == NULL) {
        NouNod->next = NouNod; *cap = NouNod; return;
    }
    // Dacă lista deja are câteva noduri
    else {
        // Mutăm primul nod în ultimul nod
        while (curent->next != *cap) {
            curent = curent->next;
        }
        // Punem prima sau adresa nodului principal în noua
adresă de nod
        NouNod->next = *cap;
        // Punem adresa nouă a nodului în ultima adresă de nod
(următor)
        curent->next = NouNod;
    }
}
// Funcție pentru a imprima lista circulară
void afisareLista(struct Nod* cap){
    struct Nod* curent = cap;
    // Dacă lista este goală, pur și simplu arată mesajul
    if (cap == NULL) {
        cout<<"\n\tLista circulara este goala!\n";
        return;
    }
    // Traversăm de la primul până la ultimul nod
    else {
```

```
        do {
            cout<<" "<<curent->data; curent = curent->next;
        } while (curent != cap);
    }
}
// Programul principal
int main(){
    struct Nod* Cap = NULL;
    cout << "Lista initiala: "; afisareLista(Cap);
    InserareNod(&Cap, 15); InserareNod(&Cap, 11);
    InserareNod(&Cap, 25); InserareNod(&Cap, 95);
    InserareNod(&Cap, 38); InserareNod(&Cap, 29);
    InserareNod(&Cap, 81); InserareNod(&Cap, 26);
    InserareNod(&Cap, 37); InserareNod(&Cap, 17);
    cout << "\nLista initiala: "; afisareLista(Cap);
    afisareMinMax(&Cap);
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Lista initiala:
        Lista circulara este goala!

Lista initiala:  15 11 25 95 38 29 81 26 37 17
        Nodul minimum = 11
        Nodul maximum = 95
```

```
Lista initiala:
        Lista circulara este goala!

Lista initiala:  211 315 216 617 715 311 518 312 818 150
        Nodul minimum = 150
        Nodul maximum = 818
```

Problema 5 – Suma și produsul nodurilor divizibile cu K

Fie avem o listă circulară simplu înlănțuită ce conține N noduri . Să se elaboreze subprograme pentru a calcula suma și produsul nodurilor listei ce sunt divizibile cu un număr K.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
int k=11;
/* Nodul listei simple */
struct Nod {
    int data;
    struct Nod* next;
};
// Funcția ce calculează suma și produsul
void sumaProdusul(struct Nod* cap, int cheia){
    struct Nod* curent = cap;
    int suma = 0, produsul = 1;
    // Dacă lista este goală, atunci va afișa un mesaj
    if (cap == NULL) {
        cout<<"\tLista este goala!\n"; return;
    }
    // Traversăm primul până la ultimul nod
    else {
        do {
            // Verificăm dacă datele nodului curent sunt diviz-
ibile prin cheie
            if ((curent->data) % cheia == 0) {
                // Calculăm suma și produsul
                suma += curent->data; produsul *= curent->data;
            }
            curent = curent->next;
        } while (curent != cap);
    }
    cout << "\nSuma numerelor divizibile cu "<<k<<" este: \t\t"
<< suma ;
    cout << "\nProdusul numerelor divizibile cu "<<k<<" este: \
t" << produsul;
```

```
}
// Funcția de afișare a listei
void afisareLista(struct Nod* cap){
    struct Nod* curent = cap;
    // Dacă lista este goală, atunci va afișa un mesaj
    if (cap == NULL) {
        cout<<"\tLista este goala!\n"; return;
    }
    //  Traversăm primul până la ultimul nod
    else {
        do {
            cout<<curent->data<<" "; curent = curent->next;
        } while (curent != cap);
    }
}
// Funcția de a insera un nod la sfârșitul unei liste circulare
void InserareNod(struct Nod** cap, int data){
    struct Nod* curent = *cap;
    // Creăm un nod nou
    struct Nod* NoulNod = new Nod;
    // Verificăm dacă nodul este creat sau nu
    if (!NoulNod) {
        cout<<"\nEroare de memorie!\n"; return;
    }
    // Introducem date în nodul nou creat
    NoulNod->data = data;
    // Verificăm dacă lista este goală, dacă nu aveți niciun
nod, atunci vom crea primul nod
    if (*cap == NULL) {
        NoulNod->next = NoulNod; *cap = NoulNod; return;
    }
    // Dacă lista are deja un careva nod
    else {
        // Mutăm primul nod în ultimul nod
        while (curent->next != *cap) {
            curent = curent->next;
        }
        // Punem primul sau adresa nodului principal în noua
legătură de nod
        NoulNod->next = *cap;
        // Punem adresa nouă a nodului în ultima legătură de
nod (următor)
```

```
        curent->next = NoulNod;
    }
}
// Programul principal
int main(){
    struct Nod* Cap = NULL;
    cout << "Lista initiala: "; afisareLista(Cap);
    // Introducem elementele listei circulare
    InserareNod(&Cap, 15); InserareNod(&Cap, 11);
    InserareNod(&Cap, 22); InserareNod(&Cap, 95);
    InserareNod(&Cap, 38); InserareNod(&Cap, 99);
    cout << "\nLista initiala: \t\t"; afisareLista(Cap);
    cout << "\nCheia de divizibilitate este: \t"<<k;
    cout << endl; sumaProdusul(Cap, k);
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Lista initiala:         Lista este goala

Lista initiala:                 15 11 22 95 38 99
Cheia de divizibilitate este:   11

Suma numerelor divizibile cu 11 este:           132
Produsul numerelor divizibile cu 11 este:       23958
```

```
Lista initiala:         Lista este goala

Lista initiala:                 15 10 22 95 38 20
Cheia de divizibilitate este:   5

Suma numerelor divizibile cu 5 este:            140
Produsul numerelor divizibile cu 5 este:        285000
```

Problema 6 – Metoda bulelor pentru sortarea unei LCSÎ

Fie avem o listă circulară simplu înlănțuită ce conține N noduri . Să se elaboreze subprograme pentru a ordona ascendent și descendent elementele unei liste circulare conform metodei bulelor.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
/* Nodul listei simple */
struct nod{
    int data;
    struct nod *next;
};
// Programul principal include metodele de sortare
int main(){
    struct nod *temp1,*temp2, *t,*NoulNod, *start;
    int n,k,i,j; start=NULL;
    cout<<"Introdu numarul de elemente ale listei: \t"; cin>>n;
    // Introducem elemente listei circulare
    cout<<"Introdu elementele listei circulare: \t\t";
    for(i=1;i<=n;i++){
        if(start==NULL){
              NoulNod=(struct nod *)malloc(sizeof(struct nod));
              cin>>NoulNod->data; NoulNod->next=NULL;
              start = NoulNod; temp1=start;
        }
        else{
              NoulNod=(struct nod *)malloc(sizeof(struct nod));
              cin>>NoulNod->data; NoulNod->next=NULL;
              temp1->next = NoulNod; temp1=NoulNod;
        }
    }
    // Metoda bulelor pentru sortarea ascendentă a listei
    cout<<"Sortarea ascendenta a listei circulare: \t";
    for(i=n-2;i>=0;i--){
       temp1=start; temp2=temp1->next;
       for(j=0;j<=i;j++){
            if(temp1->data > temp2->data){
```

```
            k=temp1->data; temp1->data=temp2->data;
                           temp2->data=k;
            }
            temp1=temp2; temp2=temp2->next;
         }
    }
    // Afișarea ascendentă a elementelor listei circulare
    t=start;
    while(t!=NULL){
       cout<<t->data<<" ";   t=t->next;
    }
    // Metoda bulelor pentru sortarea ascendentă a listei
    cout<<"\nSortarea descendenta a listei circulare: \t";
    for(i=n-2;i>=0;i--){
       temp1=start; temp2=temp1->next;
       for(j=0;j<=i;j++){
          if(temp1->data < temp2->data){
            k=temp1->data; temp1->data=temp2->data;
            temp2->data=k;
          }
          temp1=temp2; temp2=temp2->next;
       }
    }
    // Afișarea descendentă a elementelor listei circulare
    t=start;
    while(t!=NULL){
       cout<<t->data<<" ";   t=t->next;
    }
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Introdu numarul de elemente ale listei:         5
Introdu elementele listei circulare:            3 6 8 0 9
Sortarea ascendenta a listei circulare:         0 3 6 8 9
Sortarea descendenta a listei circulare:        9 8 6 3 0
```

4.5 Modele de probleme propuse

1. Elaborați subprograme în C++ pentru următoarele probleme simple.

- ***a.*** *Să se grupeze zerourile într-o listă circulară simplă.*
- ***b.*** *Să se concateneze două liste circulare astfel încât să se obțină tot o listă circulară.*
- ***c.*** *Fiind dată o listă circulară (simplu / dublu înlănțuită) de numere naturale de două cifre, să se determine câte elemente are lista respectivă.*
- ***d.*** *Să se grupeze zerourile într-o listă circulară dublă.*
- ***e.*** *Să se descompună două liste circulare în două liste circulare cu număr apropiat de elemente.*
- ***f.*** *Fiind dată o listă de numere naturale de două cifre, să se determine dacă această listă este circulară sau liniară (simplu / dublu înlănțuită).*

2. Elaborați un program în C++ pentru următoarea problemă cu fișiere.

2.1 *Să se genereze o listă circulară care prelucrează numere întregi:*
- *să se parcurgă lista începând de la primul nod în ambele sensuri;*
- *să se parcurgă lista în ambele sensuri din două în două elemente de n ori.*

2.2 *Să se genereze o listă circulară care prelucrează numere întregi:*
- *să se parcurgă lista începând de la ultimul nod în ambele sensuri;*
- *să se parcurgă n elemente din listă pornind de la primul din p în p elemente. Numărul p este citit din fișier sau introdus de la tastatură.*

2.3 *Să se genereze o listă circulară care prelucrează numere întregi:*
- *să se parcurgă lista începând de la un anumit nod k (introdus de la tastatură) în ambele sensuri;*
- *să se parcurgă n elemente din listă pornind de la un anumit nod s (introdus de la tastatură) din p în p elemente. Numărul p este introdus de la tastatură.*

3. Elaborați un program în C++ pentru următoarele probleme:

3.1 Problema lui Joseph

*Se consideră n elevi aranjați în cerc. Începând cu o poziție particulară p_0 se efectuează numărarea elevilor și în mod brutal se execută fiecare cel de-al **n**-lea elev. Când este executat un elev, cercul se închide după eliminarea acestuia. Execuția se termină când rămâne un singur elev, ultimul elev va rămâne în viață. Să se scrie un subprogram care afișează numele elevilor în ordinea în care au fost executați. Presupunând că se dorește salvarea unui anume elev, să se construiască un subprogram care să determine poziția p_0 astefel încât, dacă numărătoarea începe cu i_0, atunci elevul salvat să fie cel dorit. Să se scrie un program C++ care să testeze cele două subprograme.*

3.2 Problema cifrului

*Un cifru (cod) conține **n** marcaje (valori de la 1 la **n**). Din fișierul date.txt se citesc perechi (x y) reprezentând numărul de rotații și sensul de rotație (orar sau antiorar, unde y=1 pentru sens orar și y=2 pentru sens antiorar). Să se afișeze codul obținut.*

Exemplu:

Pentru marcajele 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 și perechile: (3 1), (4 1), (2 1), (4 2), (5 2) se obține codul: 3 6 7 4 10.

3.3 Problema Fibinacci

Având în vedere o listă circulară simplu înlănțuită care conține N noduri, sarcina este de a elimina toate nodurile din lista care conține valorile datelor Fibonacci (numerele din seria lui Fibonacci).

Exemplu:

Pentru 9⟶11⟶34⟶6⟶13⟶20, soluția va fi: 9->11->6->20.

Explicație: Lista conține două valori din seria lui Fibonacci: 32 și 13. Prin urmare, nodurile care conțin aceste date au fost șterse.

4. Probleme suplimentare pentru LCSÎ:

1. *Să se scrie programul pentru crearea unei liste circulare simplu înlănţuite cu preluarea datelor de la tastatură. Sfârşitul introducerii datelor este marcat standard. După creare, se va afişa conţinutul listei apoi se va elibera memoria ocupată.*
2. *Să se scrie funcţia pentru inserarea unui nod la începutul unei liste circulare simplu înlănţuite. Funcţia are ca parametri capul listei şi valoarea care trebuie inserată. Prin numele funcţiei se returnează noul cap al listei.*
3. *Să se scrie funcţia pentru inserarea unui nod într-o listă circulară simplu înlănţuită după un nod identificat prin valoarea unui câmp. Funcţia are ca parametri capul listei, valoarea care trebuie inserată şi valoarea după care se inserează. Dacă valoarea căutată nu este găsită, atunci inserarea se face la sfârşit.*
4. *Să se scrie funcţia pentru inserarea unui nod într-o listă circulară simplu înlănţuită înaintea unui nod identificat prin valoarea unui câmp. Funcţia are ca parametri capul listei, valoarea care se adaugă şi valoarea înaintea căreia se adaugă. Prin numele funcţiei se întoarce noul cap al listei.*
5. *Să se scrie funcţia pentru căutarea unui nod, identificat prin valoarea unui câmp, într-o listă circulară simplu înlănţuită. Funcţia are ca parametri capul listei şi valoarea căutată. Prin numele funcţiei se întoarce adresa nodului care conţine valoarea căutată (dacă sunt mai multe noduri cu aceeaşi valoare se întoarce adresa ultimului). Dacă nodul căutat nu este găsit, se întoarce valoarea NULL.*
6. *Să se scrie funcţia pentru adăugarea unui nod la sfîrşitul unei liste circulare simplu înlănţuite. Funcţia are ca parametri capul listei şi valoarea care trebuie adăugată. Prin numele funcţiei se întoarce noul cap al listei.*
7. *Să se scrie funcţia pentru ştergerea primului nod al unei liste circulare simplu înlănţuite. Funcţia are ca parametru capul listei şi întoarce noul cap al listei.*
8. *Să se scrie funcţia pentru ştergerea ultimului nod dintr-o listă circulară simplu înlănţuită. Funcţia are ca parametru capul listei şi întoarce, prin numele ei, noul cap al listei.*

4.6 Portofoliul elevului pentru LCSÎ & LCDÎ

Se consideră o listă circulară simplu (dublu) înlănțuită care memorează valori întregi. Elaborați subprograme pentru următoarele operațiuni:

a. Introduceți un număr nou în listă;
b. Afișați elementele pare și impare ale listei;
c. Afișați elementele pare pozitive și pare negative ale listei;
d. Afișați elementele impare pozitive și impare negative ale listei;
e. Afișați elementele în ordine crescătoare ale listei;
f. Afișați elementele în ordine descrescătoare ale listei;
g. Să se șteargă primul nod (ultimul nod)care conține valoarea x.
h. Să se șteargă nodurile care conțin pătrate perfecte.
i. Să se șteargă nodurile care conțin numere Fibonacci.

Fiecare elev din subgrupă va realiza o bibliotecă ce va include operațiile de mai sus, cu condiția că elementele inițiale ale listei sunt citite din fișier, fiecare elev va avea o listă individuală conform numărului de ordine.

Nr.	Secvența de numere	Nr.	Secvența de numere
1	50 31 30 35 82 85 16 18 58 61	**11**	21 37 56 32 44 71 99 87 72 90
2	79 59 45 14 57 23 73 83 51 62	**12**	19 12 49 18 37 71 39 68 42 98
3	77 81 66 84 95 43 38 65 15 76	**13**	75 46 97 40 57 70 54 73 14 59
4	26 64 39 13 17 10 40 55 29 48	**14**	56 31 44 27 96 80 81 86 58 65
5	47 74 27 91 75 88 33 25 93 42	**15**	53 21 50 67 83 69 93 32 43 17
6	67 22 69 96 46 97 28 12 41 36	**16**	41 95 89 77 29 33 87 51 99 72
7	94 63 92 34 53 78 11 52 19 86	**17**	61 76 45 24 79 85 52 92 15 36
8	70 24 89 80 60 49 68 20 98 54	**18**	82 64 74 38 23 16 63 30 10 62
9	90 60 11 78 35 84 22 88 20 34	**19**	62 12 38 79 43 90 17 23 80 92
10	55 48 28 26 66 25 91 94 13 47	**20**	30 42 50 49 41 11 71 58 83 44

4.7 Model de test grilă pentru LCSÎ & LCDÎ

Nr.	Conținutul	Punctaj
1	*Un inel reprezintă rezultatul relației de ordine dintre elementele unei liste liniare definită astfel încât ultimul element al liste are ca succesor primul element.* ◯ adevărat ◯ fals	*2p*
2	*Operațiunea de ștergere a listei circulare implică localizarea nodului care urmează să fie șters și fără eliberarea memoriei acestuia.* ◯ adevărat ◯ fals	*2p*
3	*Funcția malloc() din C++ alocă un bloc de memorie neinițializată și returnează un pointer gol (void) la primul octet al blocului de memorie alocat dacă alocarea reușește.* ◯ adevărat ◯ fals	*2p*
4	*Următoarea secvență de cod C++ prezinta reprezentarea cărui tip de listă:* `struct node{ int data; struct node *next; struct node *prev; };` ◯ LCSÎ ◯ LLSÎ ◯ LCDÎ ◯ LLDÎ	*2p*
5	*. . . este un tip de structură de date care este alcătuită din noduri care sunt create folosind structuri auto-referențiale.* ◯ LCSÎ ◯ LLSÎ ◯ LCDÎ ◯ LLDÎ	*2p*
6	*Selectați avantajele listelor circulare: LCSÎ & LCSÎ.* ☐ orice nod poate fi un punct de plecare ☐ utile pentru implementarea cozii ☐ utile în aplicații pentru parcurgeri repetate ☐ răspunsul corect lipsește	*6p*

7	*Orice unitate dintr-o listă circulară înlănțuită (legată) cu datele și indicatorul său se numește . . .* ◯ cap ◯ coadă ◯ nod ◯ pointer	*2p*
8	*Care e denumirea unității dintr-o LCÎ cu datele și indicatorul său ce definește începutul listei circulare înlănțuite ?* ◯ cap ◯ coadă ◯ nod ◯ pointer	*2p*
9	*Care este denumirea funcției din C++ STL ce returnează valoarea primului element din listă.* ◯ front() ◯ pop_front() ◯ back() ◯ push_front()	*2p*
10	*Care este denumirea funcției din C++ STL ce returnează un iterator ce indică ultimul element teoretic care urmează ultimului element.* ◯ end() ◯ pop_back() ◯ back() ◯ push_back()	*2p*
11	*Care este denumirea funcției din C++ STL ce este utilizată pentru a transfera elemente dintr-o listă în alta.* ◯ merge() ◯ emplace() ◯ splice() ◯ swap()	*2p*
12	*Denumirea funcției din C++ STL ce elimina toate elementele dintr-o listă, acelea care sunt egale cu elementul dat.* ◯ reverse() ◯ emplace() ◯ erase() ◯ remove()	*2p*

Barem de notare propus

Nota	10	9	8	7	6	5	4	3	2
%	100-95	94-88	87-78	77-63	62-48	47-33	32-21	20-10	9-5
Punctaj	28-27	26-25	24-22	21-18	19-13	12-9	8-6	5-3	2-1

5. STIVA

5.1 Introducere

Stivele sunt structuri de date logice (implementarea este făcută utilizând alte structuri de date) şi omogene (toate elementele sunt de acelaşi tip). Structura dinamică stiva are două operaţii de bază: adăugarea şi extragerea unui element. În afara acestor operaţii se pot implementa şi alte operaţii utile: test de structură vidă, obţinerea primului element fără extragerea acestuia, ş.a.m.d. Stiva foloseşte o disciplină de acces de tip LIFO (Last In First Out).

Stivele pot fi implementate în mai multe moduri. Cele mai utilizate implementări sunt cele folosind masive şi liste. Ambele abordări au aspecte pozitive şi aspecte negative:

	Aspecte pozitive	***Aspecte negative***
Masive	● *implementare simplă;* ● *consum redus de memorie;* ● *viteză mare pentru operaţii;*	■ *numărul de elemente este limitat;* ■ *risipă de memorie în cazul în care dimensiunea alocată este mult mai mare decât numărul efectiv de elemente;*
Liste	● *număr oarecare de elemente.*	■ *consum mare de memorie pentru memorarea legăturilor.*

Funcțiile asociate unei stive sunt:

- *empty () - returnează dacă stiva este goală;*
- *size () - returnează dimensiunea stivei;*
- *top () - returnează o referință la elementul cel mai de sus al stivei;*
- *push (g) - adaugă elementul „g” în partea de sus a stivei;*
- *pop () - șterge elementul cel mai înalt din stivă.*

5.2 Structura unei stive

Pentru implementarea unei stive folosind masive avem nevoie de un masiv V de dimensiune n pentru memorarea elementelor. Ultimul element al masivului va fi utilizat pentru memorarea numărului de elemente ale stivei.

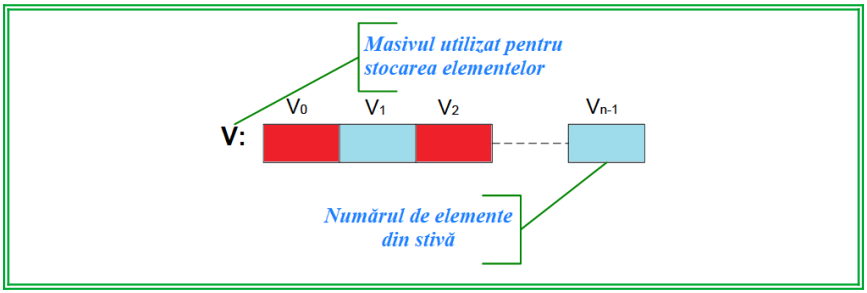


Figura 4.1 – Reprezentarea grafică a stivei utilizând masive

În cazul în care stiva este vidă, elementul V_{n-1} va avea valoarea 0. Folosind această reprezentare, operaţiile de bază se pot implementa în timp constant egal cu O(1).

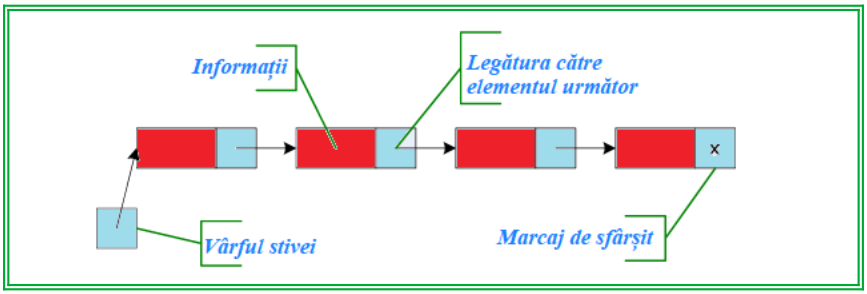


Figura 4.2 – Reprezentarea grafică a stivei utilizând lista

Fiecare nod este format din informaţiile utile şi o legătură către elementul următor. Stiva vidă este reprezentată printr-un pointer nul. Elementele sunt adăugate înaintea primului element (cu deplasarea varfului stivei). Extragerea se face tot din vărful stivei.

5.3 Operații principale efectuate

5.3.1 Inserarea (adăugarea) unui element în stivă

Algoritmul pentru implementarea operației de inserare(în pseudocod) pentru cazul în care o stivă este reprezentată printr-un masiv este:

```
adaugare(elem, V, n)
if v[n-1] = n-1      //verificam dacă stiva nu e plină
return "stiva plina"
v[v[n-1]] = elem     //adaugăm elementul în masiv
v[n-1] = v[n-1] + 1  //incrementăm numărul de elemente
return "succes"
```

5.3.2 Ștergerea (extragerea)unui element din stivă

Algoritmul pentru implementarea operației de ștergere(în pseudocod) pentru cazul în care o stivă este reprezentată printr-un masiv este:

```
stergere(V, n)
if v[n-1] = 0         //verificam dacă stiva nu e goală
return "stiva goala"
elem = v[v[n-1] - 1]  //extragem elementul din masiv
v[n-1] = v[n-1] + 1   //decrementăm numărul de elemente
return elem
```

5.3.3 Elaborare unei biblioteci în C++ pentru a opera cu stiva

Tipul informațiilor stocate în stivă este indicat de utilizator prin definirea tipului TipStiva. Codul sursă pentru biblioteca care implementează operațiile pe stiva alocată dinamic este:

```
#ifndef STIVA_H
#define STIVA_H
// Un element din stiva
struct NodStiva {
      TipStiva Date; // tipul definit de utilizator
      NodStiva *Urmator; // legatura catre elementul urmator
```

```
      // constructor pentru initializarea unui nod
      NodStiva(TipStiva date, NodStiva *urmator = NULL) :
      Date(date), Urmator(urmator){}
};
// Stiva este memorata ca un pointer catre primul element
typedef NodStiva* Stiva;
// Creaza o stiva vida
Stiva StCreare(){
      return NULL;
}
// Verivica daca o stiva este vida
bool StEGoala(Stiva& stiva){
      return stiva == NULL;
}
// Adauga un element in stiva
void StAdauga(Stiva& stiva, TipStiva date){
      stiva = new NodStiva(date, stiva);
}
// Intoarce o copie a varfului stivei
TipStiva StVarf(Stiva& stiva){
     // Caz 1: stiva vida
     if (StEGoala(stiva)) // daca stiva e goala, atunci
     return TipStiva(); // intoarcem valoarea implicita pentru
tipul stivei
     // Caz 2: stiva nevida
     return stiva->Date; // intoarcem varful stivei
}
// Extrage elementul din varful stivei
TipStiva StExtrage(Stiva& stiva){
     // Caz 1: stiva vida
     if (StEGoala(stiva)) // daca stiva e goala, atunci
     return TipStiva(); // intoarcem valoarea implicita pentru
tipul stivei
     // Caz 2: stiva nevida
      NodStiva *nodDeSters = stiva; // salvam o referinta la
nodul de sters
      TipStiva rez = stiva->Date; // salvam datele de returnat
      stiva = stiva->Urmator; // avansam capul listei
      delete nodDeSters; // stergem nodul de sters
      return rez; // intoarcem rezultatele
}
#endif //STIVA_H
```

5.3.4 Exemplu de program pentru a demonstra funcționalitatea sivei

```
#include <iostream>
#include <stack>
using namespace std;
void afisare_stiva(stack <int> s){
    while (!s.empty()){
        cout <<'\t'<< s.top();
        s.pop();
    }
    cout <<'\n';
}
int main (){
    stack <int> s;
    s.push(20); s.push(15); s.push(10);
    s.push(5); s.push(1);
    cout << "Elementele stivei sunt:\t ";
    afisare_stiva(s);
    cout << "\n s.size():\t " << s.size();
    cout << "\n s.top() :\t " << s.top();
    cout << "\n s.pop() :"; s.pop();
    afisare_stiva(s);
    return 0;
}
```

Rezultatul obținut în urma execuției secvenței de cod C++ pentru operațiile recent studiate:

```
Elementele stivei sunt:        1       5       10      15      20

 s.size():      5
 s.top() :      1
 s.pop() :      5       10      15      20
```

Remarcă:

Un alt exemplu asemănător de fișier antet pentru structura respectivă poate fi analizat în anexa 4 a acestui îndrumar.

5.4 Modele de probleme rezolvate

Problema 1 – Inversarea unui șir de caractere

Fie avem următorul mesaj: „NUME PRENUME”. Ne propunem să inversăm acest mesaj caracter cu caracter utilizând structura dinamică stiva. Să se inverseze de asemenea și un șir de numere.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;
/* Să reținem că șirul este trecut prin referință. */
void Inversare1(string &mesaj){
	stack<int> stiva; // Creăm o stivă de tip int
	// Împingem fiecare caracter din șir în stivă
	for (char caracter: mesaj) {
		stiva.push(caracter);
	}
	// Scoatem toate caracterele din stivă și le punem
înapoi în șirul de intrare
	for (int i = 0; i < mesaj.length(); i++) {
		mesaj[i] = stiva.top(); stiva.pop();
	}
}
/* Inversăm un mesaj fără a inversa cuvintele individuale*/
void Inversare2(string &mesaj){
	int jos = 0, sus = 0; // mesaj[jos..sus] este cuvânt
	stack<string> stiva; // Creăm o stivă de tip string
	// Citește mesajul
	for (int i = 0; i < mesaj.length(); i++){
	// Dacă am găsit un spațiu, atunci am găsit un cuvânt
		if (mesaj[i] == ' ') {
		   // Împingem fiecare cuvânt în stivă
		   stiva.push(mesaj.substr(jos, sus-jos+1));
	// Resetează valorile jos și sus pentru cuvântul următor
		   jos = sus = i + 1;
```

```
                }
                else { sus = i; }
        }
        stiva.push(mesaj.substr(jos)); //Împingem ultimul cuvânt
        mesaj.clear(); // Curățim mesajul
        // Umplem șirul urmând ordinea stivei
        while (!stiva.empty()){
                mesaj.append(stiva.top()).append(" ");
                stiva.pop();
        }
        mesaj.erase(prev(mesaj.end())); // Șterge ultimul spațiu
}
// Programul principal unde se implementează funcțiile create
int main(){
    // Inversarea unui șir de caractere, funcția 1
    string mesaj1 = "INFORMATICA SI TEHNOLOGII INFORMATIONALE";
    cout <<"\nMesajul 1 propus pentru inversare este: \n\t";
    cout<<mesaj1; Inversare1(mesaj1);
    cout <<"\nMesajul 1 inversat este: \n\t"<<mesaj1<<endl;
    // Inversarea unui șir de caractere, funcția 2
    string mesaj2 = "INFORMATICA & MATEMATICA";
    cout <<"\nMesajul 2 propus pentru inversare este: \n\t";
    cout<<mesaj2; Inversare2(mesaj2);
    cout <<"\nMesajul 2 inversat este: \n\t"<<mesaj2<<endl;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Mesajul 1 propus pentru inversare este:
        INFORMATICA SI TEHNOLOGII INFORMATIONALE
Mesajul 1 inversat este:
        ELANOITAMROFNI IIGOLONHET IS ACITAMROFNI

Mesajul 2 propus pentru inversare este:
        INFORMATICA & MATEMATICA
Mesajul 2 inversat este:
        MATEMATICA & INFORMATICA
```

Problema 2 – Operații fundamentale cu stiva

Să se implementeze operațiile de bază ale unei stive utilizând o listă liniară. Se vor insera minim 5 elemente, apoi se va afișa elementul din vârful stivei. Să se excludă din stivă ultimele 3 elemente și să se afișeze lista de elemente-le rămase în stivă.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
// Un nod al listei liniare
struct Nod{
	int data;
	struct Nod* next;	// pointer către următorul nod
};
// Adăugăm un element x în stivă (la început)
void push(struct Nod **varf, int x){
	struct Nod* nod = NULL; // Alocăm noul nod în heap
	nod = (struct Nod*)malloc(sizeof(struct Nod));
	// Verificăm dacă stiva este plină
	if (!nod){
	 cout<<"Supraincarcarea heap-ului!"<<endl; exit(1);
	}
	//Inserăm nodul x, set[m datele în nodul alocat
	nod->data = x;
	nod->next = *varf; // Setăm indicatorul .next al noului
nod pentru a indica nodul curent de sus al listei
	*varf = nod; // Actualizăm indicatorul de sus
}
// Funcția verifică dacă stiva este goală
int EsteGoala(struct Nod* varf){
	return varf == NULL;
}
// Funcția afișează elementul din vârful stivei
int peek(struct Nod *varf){
	if (!EsteGoala(varf)) return varf->data;
	else exit(EXIT_FAILURE);
}
// Funcția pentru a scoate elementul de sus din stivă (pop)
```

```
void pop(struct Nod** varf){
       struct Nod *nod; // Verificăm prezența fluxului de stivă
       if (*varf == NULL){
              cout<<"\n Debordarea stivei!"; exit(1);
       }
       cout<<"\tExcludem nodul "<< peek(*varf)<<endl;
       nod = *varf;
       *varf = (*varf)->next; // Actualizăm indicatorul de sus
pentru a indica următorul nod
       free(nod); // Memorie alocată gratuită (în surplus)
}
// Funcția afișează elementele stivei
void afisare(struct Nod* varf){
    Nod *c; c=varf;
    while(c){
        cout<<c->data<<" "; c=c->next;
    }
}
int main(){
       struct Nod *varf = NULL;
       if (EsteGoala(varf)) cout<<"Stiva este goala!"<<endl;
       else cout<<"Stiva nu este goala!"; push(&varf, 2);
       push(&varf, 5); push(&varf, 9); push(&varf, 7);
       cout<<"Structura stivei initiale: \t"; afisare(varf);
       cout<<"\nElementul din varful stivei este:";
       cout<<peek(varf)<<endl; pop(&varf); pop(&varf);
       pop(&varf); cout<<"Structura stivei dupa excluderea ul-
timelor 3 elemente: \n\t"; afisare(varf); return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Stiva este goala!
Structura stivei initiale:      7 9 5 2
Elementul din varful stivei este:7
        Excludem nodul 7
        Excludem nodul 9
        Excludem nodul 5
Structura stivei dupa excluderea ultimelor 3 elemente:
        2
```

Problema 3 – Suma, produsul, minim și maxim în stivă

Să se creeze o listă liniară cu numere întregi folosind o stivă. Să se elaboreze subprograme pentru: afișarea conținutului stivei; afișarea elementului maximal și minimal al stivei; afișarea sumei și a produsului elementelor stivei.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
// Un nod al listei liniare
struct lista{
    int info;
    lista *leg;
};
lista *p, *prim, *ultim;
int n;
// Elaborăm o funcție pentru crearea unei liste (stiva)
void creare(lista *&prim, lista *&ultim){
    int i,inf;
    cout<<"Introduceti numarul de elemente a listei, n = ";
    cin>>n;
    cout<<"Introduceti primul element al stivei: ";
    cin>>inf; // primul element al stivei
    cout<<"Introduceti cele "<<n-1<<" elemente:\n";
    prim=new lista; prim->info=inf; prim->leg=NULL; ultim=prim;
    for(i=2;i<=n;i++){
        cout<<"\tIntroduceti elementul al "<<i<<"-lea: \t";
        cin>>inf;
        p=new lista; p->info=inf; p->leg=prim; prim=p;
    }
}
// Elaborăm o funcție pentru afișarea unei liste (stiva)
void afisare(lista *prim){
    p=prim;
    while(p!=NULL){
        cout<<p->info<<" "; p=p->leg;
    }
}
// Elaborăm o funcție pentru afișarea sumei elementor stivei
```

```
int suma(lista *prim){
    int suma=0; p=prim;
    while(p!=NULL){
        suma+=p->info; p=p->leg;
    }
    return suma;
}
// Elaborăm o funcție pentru afișarea produsuli elementor sti-
vei
int produsul(lista *prim){
    int produsul=1; p=prim;
    while(p!=NULL){
        produsul*=p->info; p=p->leg;
    }
    return produsul;
}
// Elaborăm o funcție pentru afișarea elementului maximal al
stivei
int maxim(lista *prim){
    int maxv= -10000; p=prim;
    while(p!=NULL){
        if( p->info > maxv ) maxv=p->info;
        p=p->leg;
    }
    return maxv;
}
// Elaborăm o funcție pentru afișarea elementului minimal al
stivei
int minim(lista *prim){
    int minv= INT_MAX; p=prim;
    while(p!=NULL){
        if( p->info < minv ) minv=p->info;
        p=p->leg;
    }
    return minv;
}
// Programul principal
int main(){
    creare(prim, ultim);
    cout<<"Stiva compusa din cele "<<n<<" elemente este:\n\t";
    afisare(prim);
    cout<<endl<<"\nSuma elementelor este:\t\t"<<suma(prim);
```

```
    cout<<endl<<"Produsul elementelor este:\t"<<produsul(prim);
    cout<<endl<<"Elementul Maximal:\t"<<maxim(prim);
    cout<<endl<<"Elementul Minimal:\t"<<minim(prim)<<endl;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Introduceti numarul de elemente a listei, n = 5
Introduceti primul element al stivei: 1
Introduceti cele 4 elemente:
        Introduceti elementul al 2-lea:         2
        Introduceti elementul al 3-lea:         3
        Introduceti elementul al 4-lea:         4
        Introduceti elementul al 5-lea:         5
Stiva compusa din cele 5 elemente este:
        5 4 3 2 1

Suma elementelor este:          15
Produsul elementelor este:      120
Elementul Maximal:      5
Elementul Minimal:      1
```

```
Introduceti numarul de elemente a listei, n = 7
Introduceti primul element al stivei: -12
Introduceti cele 6 elemente:
        Introduceti elementul al 2-lea:         24
        Introduceti elementul al 3-lea:         13
        Introduceti elementul al 4-lea:         6
        Introduceti elementul al 5-lea:         9
        Introduceti elementul al 6-lea:         5
        Introduceti elementul al 7-lea:         -19
Stiva compusa din cele 7 elemente este:
        -19 5 9 6 13 24 -12

Suma elementelor este:          26
Produsul elementelor este:      19206720
Elementul Maximal:      24
Elementul Minimal:      -19
```

Problema 4 – Numărul de elemente pare, impare și prime

Să se creeze o listă liniară cu numere întregi folosind o stivă. Să se elaboreze subprograme pentru: afișarea conținutului stivei; afișarea tuturor elementelor pare, impare și prime dintr-o stivă.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
// Un nod al listei liniare
struct lista{
    int info;
    lista *leg;
};
lista *p, *prim, *ultim;
int n;
// Elaborăm o funcție pentru crearea unei liste (stiva)
void creare(lista *&prim, lista *&ultim){
    int i,inf;
    cout<<"Introduceti numarul de elemente a listei, n = ";
    cin>>n;
    cout<<"Introduceti primul element al stivei: ";
    cin>>inf; // primul element al stivei
    cout<<"Introduceti cele "<<n-1<<" elemente:\n";
    prim=new lista; prim->info=inf; prim->leg=NULL; ultim=prim;
    for(i=2;i<=n;i++){
        cout<<"\tIntroduceti elementul al "<<i<<"-lea: \t";
         cin>>inf;
        p=new lista; p->info=inf; p->leg=prim; prim=p;
    }
}
// Elaborăm o funcție pentru afișarea unei liste (stiva)
void afisare(lista *prim){
    p=prim;
    while(p!=NULL){
        cout<<p->info<<" "; p=p->leg;
    }
}
// Funcția pentru a determina numărul de divizori ai nodului
```

```
int divizori(int x){
    int q=1;
    for(int i=2;i<=x/2;i++)
    if(x%i==0) q=0;
    return q;
}
// Elaborăm o funcție pentru afișarea numărului de elemente
pare ale stivei
int pare(lista *prim){
    int nr1=0; p=prim;
    while(p!=NULL){
        if(p->info %2==0) nr1++; p=p->leg;
    }
    return nr1;
}
// Elaborăm o funcție pentru afișarea numărului de elemente im-
pare ale stivei
int impare(lista *prim){
    int nr2=0; p=prim;
    while(p!=NULL){
        if(p->info %2!=0) nr2++; p=p->leg;
    }
    return nr2;
}
// Elaborăm o funcție pentru afișarea numărului de elemente
prime ale stivei
int prime(lista *prim){
    int nr=0; p=prim;
    while(p!=NULL){
        if(divizori(p->info)==1) nr++; p=p->leg;
    }
    return nr;
}
// Programul principal
int main(){
    creare(prim, ultim);
    cout<<"Stiva compusa din cele "<<n<<" elemente este:\t\t";
    afisare(prim);
    cout<<"\nNumarul de elemente pare din stiva:\t\t";
    cout<<pare(prim);
    cout<<"\nNumarul de elemente impare din stiva:\t\t";
    cout<<impare(prim);
```

```
    cout<<"\nNumarul de elemente prime din stiva:\t\t";
    cout<<prime(prim);
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Introduceti numarul de elemente a listei, n = 5
Introduceti primul element al stivei: 5
Introduceti cele 4 elemente:
        Introduceti elementul al 2-lea:         7
        Introduceti elementul al 3-lea:         9
        Introduceti elementul al 4-lea:         10
        Introduceti elementul al 5-lea:         12
Stiva compusa din cele 5 elemente este:         12 10 9 7 5
Numarul de elemente pare din stiva:             2
Numarul de elemente impare din stiva:           3
Numarul de elemente prime din stiva:            2
```

```
Introduceti numarul de elemente a listei, n = 8
Introduceti primul element al stivei: 9
Introduceti cele 7 elemente:
        Introduceti elementul al 2-lea:         8
        Introduceti elementul al 3-lea:         7
        Introduceti elementul al 4-lea:         6
        Introduceti elementul al 5-lea:         5
        Introduceti elementul al 6-lea:         4
        Introduceti elementul al 7-lea:         3
        Introduceti elementul al 8-lea:         2
Stiva compusa din cele 8 elemente este:         2 3 4 5 6 7 8 9
Numarul de elemente pare din stiva:             4
Numarul de elemente impare din stiva:           4
Numarul de elemente prime din stiva:            4
```

Problema 5 – Transformări ale șirurilor de caractere

Să se creeze o listă liniară cu elemente litere mici folosind o stivă. Să se elaboreze subprograme pentru:

- *afișarea conținutului stivei;*
- *înlocuirea fiecărei vocale cu litera imediat următoare din alfabet și afișarea stivei modificate;*
- *să se transforme fiecare literă a stivei în literă majusculă și să se afișeze acest rezultat.*

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
#include <string>
using namespace std;
// Un nod al listei liniare
struct lista{
    char info;
    lista *leg;
}; lista *p, *prim, *ultim;
int n;
// Elaborăm o funcție pentru crearea unei liste (stiva)
void creare(lista *&prim, lista *&ultim){
    int i; char inf;
    cout<<"Introduceti numarul de elemente ale listei, n = ";
    cin>>n;
    cout<<"Introduceti primul element al stivei: "; cin>>inf;
    cout<<"Introduceti cele "<<n-1<<" elemente:\n";
    prim=new lista; prim->info=inf; prim->leg=NULL; ultim=prim;
    for(i=2;i<=n;i++){
        cout<<"\tIntroduceti elementul al "<<i<<"-lea: \t";
        cin>>inf; p=new lista; p->info=inf; p->leg=NULL;
        ultim->leg=p; ultim=p;
    }
}
// Elaborăm o funcție pentru afișarea unei liste (stiva)
void afisare(lista *prim){
    p=prim;
    while(p!=NULL){
        cout<<p->info<<" "; p=p->leg;
```

```
    }
}
// Elaborăm o funcție pentru a înlocui vocalele cu următoare
literă din alfabet
void inlocuire(lista *prim){
    p=prim; char voc[10]="aeiou";
    while(p!=NULL){
        for(int i=0;i<n;i++)
            if(voc[i]==p->info) p->info=p->info+1; p=p->leg;

    }
    cout<<endl<<"Stiva obtinuta dupa inlocuire: \t\t\t";
    afisare(prim);
}
// Elaborăm o funcție pentru a transforma în majuscule elemente
stivei
void transformare(lista *prim){
    p=prim;
    while(p!=NULL){
        p->info=p->info-32; p=p->leg;
    }
    cout<<endl<<"Stiva obtinuta dupa transformare: \t\t";
    afisare(prim);
}
int main(){
    creare(prim, ultim);
    cout<<"Stiva compusa din cele "<<n<<" elemente este: \t";
    afisare(prim); inlocuire(prim); transformare(prim);
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Introduceti numarul de elemente ale listei, n = 4
Introduceti primul element al stivei: m
Introduceti cele 3 elemente:
        Introduceti elementul al 2-lea:         a
        Introduceti elementul al 3-lea:         m
        Introduceti elementul al 4-lea:         a
Stiva compusa din cele 4 elemente este:         m a m a
Stiva obtinuta dupa inlocuire:                  m b m b
Stiva obtinuta dupa transformare:               M B M B
```

Problema 6 – Transportare containere

Se consideră o stivă de containere cu caracteristicile: cod container (un număr cuprins între 1 și 99), conținut (un câmp alfanumeric de 20 caractere), data ambalării (zi, luna și an) si greutatea (în kg). Să se simuleze activitățile de stivuire și de încărcare într-un mijloc de transport a containerelor din stivă.

Rezolvare:

Evenimentele care pot avea loc cu această stivă de containere sunt: sosirea unui nou container pentru stivuire simulată prin apăsarea tastei S și încărcarea într-un mijloc de transport a unui container simulată prin apăsarea tastei I.

Pentru un container sosit pentru stivuire se vor cere: codul, conținutul, data ambalării și greutatea, iar la încărcarea sa într-un mijloc de transport pentru plecare se afișează aceleași informații. Stiva containerelor se va implementa ca o listă simplu înlănțuită, identificată prin variabila vârf, care arată containerul vizibil din vârful stivei, adică primul element din lista de containere (ultimul așezat în stivă).

Funcția push() va simula așezarea unui nou container în stivă și va returna valoarea 0 în cazul în care nu s-a reușit alocarea dinamică a spațiului de memorie necesar pentru introducerea unui nou element (informațiile containerului) în stiva de containere și valoarea 1 în caz contrar. Functia pop() va oferi informații, prin intermediul parametrilor săi, despre containerul scos din stivă pentru expediere și va returna valoarea 1 dacă în stivă nu mai sunt containere și valoarea 0 în caz contrar.

Prezentarea soluției problemei în limbajul C++

```
#include<iostream>
#include<string.h>
#include<stdlib.h>
#include<conio.h>
using namespace std;
```

```
struct tipdata{
    int zi, luna, an;
};
struct tipstiva{
    int cc, greutate;
    char continut[20];
    struct tipdata data;
    struct tipstiva *urm;
};
struct tipstiva *varf;
/* descrierea functiei push() */
int push(int *wcc,char *wcontinut,struct tipdata *wdata,int
*wgreutate){
    struct tipstiva *ptrcontainer;
    ptrcontainer=(struct tipstiva*)malloc(sizeof(struct tip-
stiva));
    if(ptrcontainer==NULL){
        cout<<"\n Memorie insuficienta pentru memorarea unui
container!";
        return 0;
    }
    ptrcontainer->cc=*wcc;
    strcpy(ptrcontainer->continut,wcontinut);
    ptrcontainer->data.zi=wdata->zi;
    ptrcontainer->data.luna=wdata->luna;
    ptrcontainer->data.an=wdata->an;
    ptrcontainer->greutate=*wgreutate;
    ptrcontainer->urm=varf; varf=ptrcontainer;
    return 1;
}
/* descrierea functiei de extragere din stiva */
int pop(int *ptrcc,char *ptrcontinut,struct tipdata
*ptrdata,int *ptrgreutate){
    struct tipstiva *ptrcontainer;
    ptrcontainer=varf;
```

```
    if(!varf) return 0;
    varf=ptrcontainer->urm; *ptrcc=ptrcontainer->cc;
    strcpy(ptrcontinut,ptrcontainer->continut);
    ptrdata->zi=ptrcontainer->data.zi;
    ptrdata->luna=ptrcontainer->data.luna;
    ptrdata->an=ptrcontainer->data.an;
    *ptrgreutate=ptrcontainer->greutate;
    free(ptrcontainer); return 1;
}
// Programul principal
int main(){
    int wcc, wgreutate, terminat=0;
    char wcontinut[20], rasp;
    struct tipdata wdata;
    varf=NULL;
    while(!terminat){
        system("cls");
        cout<<"\n A - asezare container in stiva";
        cout<<"\n S - scoatere container din stiva";
        cout<<"\n T - terminare program";
        cout<<"\n Alegeti operatia dorita:";
        rasp=getche();
        switch (rasp){
        case 'A': cout<<"\n cod container:"; cin>>wcc;
                  cout<<"\n continut container:";
                  cin>>wcontinut;
                  cout<<"\n ziua ambalarii:"; cin>>wdata.zi;
                  cout<<"\n luna ambalarii:"; cin>>wdata.luna;
                  cout<<"\n anul ambalarii:"; cin>>wdata.an;
                  cout<<"\n greutate container:";
                  cin>>wgreutate;
                  if(push(&wcc,wcontinut,&wdata,&wgreutate))
                  cout<<"\n Containerul a fost asezat in
stiva.";
                  else cout<<"\n Memorie insuficienta pentru
```

```
container.";
                    break;
        case 'S': if(pop(&wcc,wcontinut,&wdata,&wgreutate))
                    cout<<"\n Containerul "<<wcc<<"
"<<wcontinut<<" "<<wgreutate<<" din "<<wdata.zi<<" "<<wda-
ta.luna<<" "<<wdata.an;
                    else cout<<"\n Nu mai exista containere";
                    break;
        case 'T': terminat=1; break;
        default:  cout<<"\n Operatie aleasa gresit";
        }
        cout<<"\n Apasati o tasta pentru continuare:";
        getch();
    }
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
 A - asezare container in stiva
 S - scoatere container din stiva
 T - terminare program
 Alegeti operatia dorita:
```

Spre exemplu dorim să așezăm un container, vom tasta opțiunea A și vom introduce datele (cod=22, conținut=banane, ziua=22, luna=05, anul=2020, greutate=500. După introducerea datelor, dorim să ștergem acest container, tastăm opțiunea S și vom vedea lista tuturor containerelor:

```
Alegeti operatia dorita:S
Containerul 22 banane 500 din 22 5 2020
```

5.5 Modele de probleme propuse

1. Elaborați subprograme în C++ pentru următoarele probleme simple.

- ***a.*** *Să se scrie funcţia de inversare a unui şir folosind o stivă alocată dinamic.*
- ***b.*** *Să se scrie funcţia de conversie a unei stive în listă simplu înlănţuită.*
- ***c.*** *Să se scrie funcţia de adăugare a unui element pentru o stivă alocată ca vector.*
- ***d.*** *Să se scrie funcţia de extragere a unui element dintr-o stivă alocată ca vector.*

2. Elaborați un program în C++ pentru următoarea problemă cu fișiere.

- ***a.*** *Într-o stivă sunt memorate numere din intervalul [a, b]. Să se elimine din acest interval fiecare număr pătrat perfect și fiecare număr prim, să se scrie rezultatul în fișier.*
- ***b.*** *Într-o stivă sunt memorate caractere (litere mici și cifre). Să se elimine din stivă fiecare vocală și număr par, apoi rezultatul să fie scris în fișier.*

3. Elaborați un program în C++ pentru următoarele probleme.

3.1 *Se consideră două şiruri de numere întregi. Să se scrie funcția de concatenare a două stive alocate dinamic S_1 şi S_2 folosind doar operațiile de bază (adăugarea, extragerea și testarea stivei vide). Rezultatul final trebuie să conțină elementele din cele două stive în ordinea inițială.* ***Indicaţie:*** *Se va folosi o stivă temporară.*

3.2 *Se consideră un şir de numere întregi. Să se scrie funcția care construieşte două stive (una cu numerele negative şi una cu cele pozitive), folosind doar structuri de tip stivă.*
Indicaţie: *Se adaugă toate elementele într-o stivă temporară după care se extrag elementele din aceasta şi se introduc în stiva corespunzătoare.*

3.3 *Se dă un labirint dreptunghiular de dimensiuni NxM, unde N, M sunt mai mici decât 50. Pozitiile din stânga sus și din dreapta jos sunt marcate cu 0, celelalte conțin unul dintre numerel 1, 2, 3, 4. Scopul este de a parcurge labirintul din colțul stânga sus până la colțul din dreapta jos, pe un drum de lungime minimă, pe direcții paralele cu laturile sale. Drumul urmat trebuie să plece din 0 în 1, apoi din 1 în 2, din 2 în 3, din 3 în 4, din 4 în 1 ș.a.m.d. Se poate ajunge în poziția finală din oricare poziție vecină. Din fișierul de intrare* ***labirint.in*** *se vor citi de pe prima linie numerele întregi N și M, care reprezintă dimensiunile labirintului, iar de pe următoarele N linii câte M numere întregi, separate prin câte un spațiu, reprezentând labirintul. În fișierul de ieșire* ***labirint.out*** *se va afișa pe prima linie numărul minim de pași, iar pe următoarea linie, un șir de caractere ce reprezintă succesiunea de mișcări de pe cel mai scurt drum din labirint folosind codificarea: D(jos), U(sus), L(stânga), respectiv R(dreapta).*

Labirint.in	Labirint.out
5 4 0 1 2 3 3 2 1 4 4 1 2 1 1 4 3 2 2 3 4 0	7 RRRDDDD

4. Elaborați o bibliotecă în C++ pentru următoarea problemă.

Într-o stivă sunt memorate numere din intervalul [a, b]. Elaborați subprograme pentru: adăugarea numerelor, ștergerea numerelor , afișarea numerelor pare, impare, prime, divizibile la un număr k citit din fișier.

- ***a.*** *Stiva este reprezentată sub forma unui masiv.*
- ***b.*** *Stiva este reprezentată sub forma unei liste simplu înlănțuite.*

5.6 Portofoliul elevului pentru stivă

Se consideră o stivă ce memorează valori întregi pozitive. Elaborați subprograme pentru următoarele operațiuni:

a. Introduceți un număr nou în stivă;
b. Afișați elementele pare și impare ale stivei;
c. Afișați elementele pare pozitive și pare negative ale stivei;
d. Afișați elementele impare pozitive și impare negative ale stivei;
e. Afișați elementele în ordine crescătoare ale stivei;
f. Afișați elementele în ordine descrescătoare ale stivei;
g. Să se șteargă primul nod (ultimul nod)care conține valoarea x.
h. Să se șteargă nodurile care conțin pătrate perfecte.
i. Să se șteargă nodurile care conțin numere Fibonacci.

Fiecare elev din subgrupă va realiza o bibliotecă ce va include operațiile de mai sus, cu condiția că elementele inițiale ale stivei sunt citite din fișier, fiecare elev va avea o stivă individuală conform numărului de ordine.

Nr.	Secvența de numere	Nr.	Secvența de numere
1	50 31 30 35 82 85 16 18 58 61	**11**	21 37 56 32 44 71 99 87 72 90
2	79 59 45 14 57 23 73 83 51 62	**12**	19 12 49 18 37 71 39 68 42 98
3	77 81 66 84 95 43 38 65 15 76	**13**	75 46 97 40 57 70 54 73 14 59
4	26 64 39 13 17 10 40 55 29 48	**14**	56 31 44 27 96 80 81 86 58 65
5	47 74 27 91 75 88 33 25 93 42	**15**	53 21 50 67 83 69 93 32 43 17
6	67 22 69 96 46 97 28 12 41 36	**16**	41 95 89 77 29 33 87 51 99 72
7	94 63 92 34 53 78 11 52 19 86	**17**	61 76 45 24 79 85 52 92 15 36
8	70 24 89 80 60 49 68 20 98 54	**18**	82 64 74 38 23 16 63 30 10 62
9	90 60 11 78 35 84 22 88 20 34	**19**	62 12 38 79 43 90 17 23 80 92
10	55 48 28 26 66 25 91 94 13 47	**20**	30 42 50 49 41 11 71 58 83 44

5.7 Model de test grilă pentru stivă

Nr.	Conţinutul	Punctaj
1	*Pentru implementarea stivei prin intermediul unui array, putem spune că un dezavantaj ar fi numărul de elemente limitat.* ◯ adevărat ◯ fals	*2p*
2	*Pentru implementarea stivei prin intermediul unui array, putem spune că un avantaj ar fi consumul redus de memorie.* ◯ adevărat ◯ fals	*2p*
3	*Funcția calloc() din C++ alocă un bloc de memorie neinițializată și returnează un pointer gol (void) la primul octet al blocului de memorie alocat dacă alocarea reușește.* ◯ adevărat ◯ fals	*2p*
4	*Stiva este o structură de date liniara ce lucrează cu o colecție de date având câteva operații principale:* ☐ push() ☐ pop() ☐ peek() ☐ răspunsul corect lipsește	*6p*
5	*Care sunt modalitățile de implementare a structurii dinamice stiva?* ☐ array ☐ folosirea containerului stack ☐ LLSÎ ☐ LCSÎ & LCDÎ	*6p*
6	*Care este denumirea funcției din C++ ce returnează valoarea elementului din listă ce a fost adăugat la urmă?* ◯ push() ◯ pop() ◯ back() ◯ push()	*2p*
7	*Care este denumirea funcției din C++ STL ce este utilizată pentru a transfera elemente dintr-o listă în alta.* ◯ abstractă ◯ exogenă ◯ neomogenă ◯ eterogenă	*2p*

8	*Care este denumirea funcției din C++ ce returnează un pointer către memoria nou alocată, care este aliniată în mod adecvat pentru orice tip de variabilă și poate fi diferită de **ptr** sau **NULL** dacă solicitarea eșuează?* ◯ realloc() ◯ calloc() ◯ malloc() ◯ free()	*2p*
9	*Care sunt avantajele implementării stivei prin mulțimi (array-uri) ?* ☐ Implementare simplă ☐ consum redus de memorie ☐ viteza mare pentru operații ☐ un număr oarecare de elemente	*6p*
10	*Se consideră o stivă în care inițial au fost introduse, în această ordine, valorile 1 și 2. Dacă se notează cu PUSH(x) operația prin care se adaugă valoarea x în vârful stivei, și POP operația prin care se extrage elementul din vârful stivei, care este conținutul acesteia în urma operațiilor:* ***POP; PUSH(3); POP; PUSH(4); PUSH(5)?*** ◯ 5 4 3 ◯ 2 3 5 ◯ 5 4 1 ◯ 1 4 5	*5p*
11	*Se consideră următoarele operații aplicate asupra unei stive vide:* • *push(s,3); push(s,1); el←pop(s); push(s,2);* • *push(s,4); el ← pop(s); el ← pop(s);* *Ce va conține la final stiva și variabila el?* ◯ s=[1,3] & el=4 ◯ s=[1] & el=4 ◯ s=[3] & el=2 ◯ s=[1] & el=2	*4p*

Barem de notare propus

Nota	10	9	8	7	6	5	4	3	2
%	100-95	94-88	87-78	77-63	62-48	47-33	32-21	20-10	9-5
Punctaj	39-37	36-34	33-30	29-25	24-19	18-13	12-8	7-4	3-2

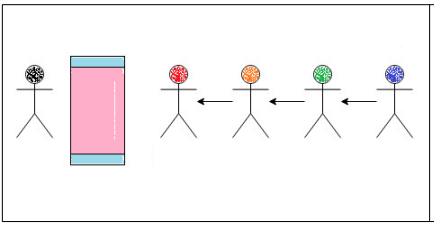
6. COADA

6.1 Introducere

Cozile sunt structuri de date logice (implementarea este făcută utilizând alte structuri de date) şi omogene (toate elementele sunt de acelaşi tip). Structura dinamică coada are două operaţii de bază: adăugarea şi extragerea unui element. În afara acestor operaţii se pot implementa şi alte operaţii utile: test de structură vidă, obţinerea primului element fără extragerea acestuia ș.a.m.d.

Diferența fundamentală între stivă și coadă este disciplina de acces. Stiva folosește o disciplină de acces de tip LIFO (Last In First Out), despre aceasta am menționat atunci când am studiat stiva, iar coada - o disciplină de acces de tip FIFO (First In First Out). Cozile pot fi implementate în mai multe moduri. Cele mai utilizate implementări sunt cele folosind masive şi liste. Ambele abordări au aspecte pozitive şi aspecte negative.

Exemplu din viața cotidiană:



- *Există un ghișeu unde vin oamenii să procure bilete și apoi pleacă cu ele la muzeu.*
- *Oamenii intră pe o linie (coadă) pentru a ajunge la Contorul de bilete într-o manieră organizată.*

În acest exemplu, trebuie luate în considerare următoarele lucruri: persoana care va intra mai întâi la coadă, va primi biletul întâi și va părăsi coada; persoana care intră la coadă va primi biletul după persoana din fața sa; în acest fel, persoana care intră ultima în coadă va primi ultima acest bilet. Prin urmare, prima persoană care intră la coadă primește biletul întâi și ultima persoană care intră în coadă primește biletul ultimul.

6.2 Structura unei cozi

Coada se poate implementa folosind un vector circular de dimensiune n. Ultimele două elemente conţin indicii de start şi sfârşit ai cozii, iar antepenultimul element este un marcaj folosit pentru a putea diferenţia cazurile de coadă goală şi coadă plină.

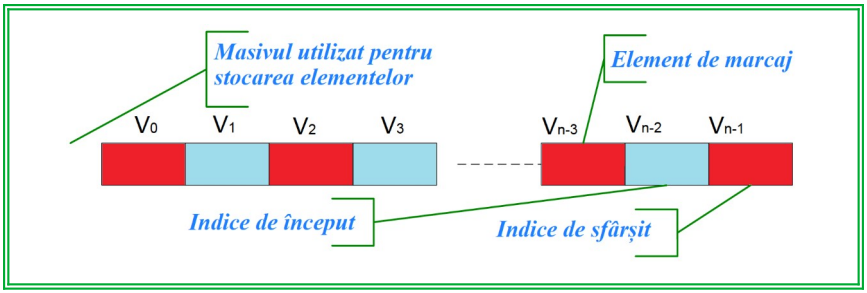


Figura 5.1 – Reprezentarea grafică a cozii utilizând masive

Cea de-a doua modalitate de implementare a cozilor este cea folosind liste alocate dinamic. Coada poate fi implementată folosind o listă circulară dublu înlanţuită.

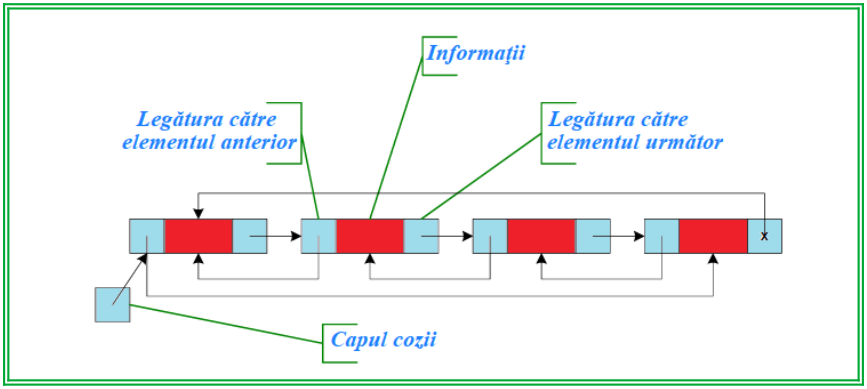


Figura 5.2 – Reprezentarea grafică a cozii utilizând o listă circulară dublă

6.3 Operații principale efectuate

6.3.1 Inserarea (adăugarea) unui element în coadă

Algoritmul pentru implementarea operației de inserare(în pseudocod) pentru cazul în care o coadă este reprezentată printr-un masiv este:

```
adaugare(elem, V, n)
v[n-2] = (v[n-2] + 1) mod (n-2)   //deplasăm capul cozii
if v[n-1] = v[n-2]                //verificare coada plină
return "coada plina"
V[V[n-1]] = elem                  //adăugăm elementul
return "succes"
```

6.3.2 Ștergerea (extragerea)unui element din coadă

Algoritmul pentru implementarea operației de ștergere(în pseudocod) pentru cazul în care o coadă este reprezentată printr-un masiv este:

```
stergere(V,n)
if v[n-1] = v[n-2]              //verificare coadă goală
return "coada goala"
v[n-1] = (v[n-1] + 1) mod (n-2) //deplasăm indicele de sfârșit
return V[V[n-1]]                //întoarcem elementul
```

6.3.3 Exemplu de program pentru a implementa coada cu C++ STL

```
#include <iostream>
#include <queue>
using namespace std;
// afisarea elementelor cozii
void afisare_coada(queue<int> q){
    while (!q.empty()){
        cout << q.front() << " "; q.pop();
    }
    cout << endl;
}
int main(){
    queue<int> q ;
```

```
    // atasarea elementelor {0, 1, 2, 3, 4} in coada
    for (int i = 0; i < 5; i++)
        q.push(i);
    // afisarea elementelor cozii
    cout << "Elementele cozii sunt: \t\t";
    afisare_coada(q);
    // Eliminarea capului cozii
    // In acest caz, cel mai vechi element, 0, va fi eliminat
    int elimin = q.front();
    q.pop();
    cout << "A fost eliminat elementul: \t" << elimin << endl;
    // afisarea elementelor cozii dupa eliminarea lui 0
    cout << "Elementele cozii sunt: \t\t";
    afisare_coada(q);
    // Vizualizarea capului cozii
    int cap = q.front();
    cout << "Capul cozii este: \t\t" << cap << endl;
    // Vizualizarea dimensiunii cozii
    int dimensiune = q.size();
    cout << "Dimensiunea cozii: \t\t" << dimensiune <<endl;
    return 0;
}
```

Rezultatul obținut în urma execuției secvenței de cod C++ pentru operațiile recent studiate:

```
Elementele cozii sunt:          0 1 2 3 4
A fost eliminat elementul:      0
Elementele cozii sunt:          1 2 3 4
Capul cozii este:               1
Dimensiunea cozii:              4
```

Remarcă:

- *Un exemplu de fișier antet pentru structura respectivă poate fi analizat în anexa 4 a acestui îndrumar.*
- *De asemenea la dorință se poate de îmbunătățit acest fișier antet cu alte subprograme pe care le veți aplica la rezolvarea problemelor.*

6.4 Modele de probleme rezolvate

Problema 1 – Implementarea cozii utilizând stiva

Fie avem o structură de date de tip stivă, cu operații push și pop. Trebuie să implementăm o coadă folosind instanțe ale structurii de date de stiva și operații pe aceasta.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
#include <stack>
using namespace std;
struct coada {
    stack<int> s1, s2;
    // Adăugați un element x în spatele cozii
    void Pozitionare(int x){
        // Mutați toate elementele de la s1 la s2
        while (!s1.empty()) {
            s2.push(s1.top()); s1.pop();
        }
        s1.push(x); // Împingeți elementul în s1
        // Împingeți totul înapoi la s1
        while (!s2.empty()) {
            s1.push(s2.top()); s2.pop();
        }
    }
    // Eliminați un element din fața cozii
    int Decuplare()    {
        // Dacă prima stivă este goală
        if (s1.empty()) {
            cout << "Stiva este goala!"; exit(0);
        }
        // Reveniți în partea de sus a lui s1
        int x = s1.top();
        s1.pop(); return x;
    }
};
// Programul principal
```

```
int main(){
    coada A;
    // Datele inițiale ale stivei
    A.Pozitionare(3); A.Pozitionare(5); A.Pozitionare(7);
    // Datele cozii după implementarea stivei
    cout<<"Solutia este: \t";
    cout<<A.Decuplare()<<" "<<A.Decuplare();
    cout<<" "<<A.Decuplare()<<endl;
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Solutia este:   3 5 7
```

```
Solutia este:   3 2 5 4 7 6 9 8 11 10
```

```
Solutia este:   13 22 35 44 57 66 79 88 90
```

Observație:
Această metodă asigură faptul că cel mai „vechi” element introdus este întotdeauna în partea de sus a stivei 1, astfel încât operațiunea Decuplare apare doar din stiva 1. Pentru a pune elementul în partea de sus a stivei 1, se folosește stiva 2.

Remarcă:

Complexitatea timpului:
- *Operațiunea push: O (n).*
- *Operațiunea pop: O (1).*

Spațiu auxiliar: O (n). Utilizarea stivei pentru stocarea valorilor.

Problema 2 – Implementarea cozii circulare

Coada circulară este o structură de date liniară în care operațiile sunt efectuate pe baza principiului FIFO (First In First Out, adică primul intrat - primul ieșit) și ultima poziție este conectată înapoi la prima poziție pentru a face un cerc. Se mai numește „inel amortizator”.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
using namespace std;
struct Coada{
    // Inițializați fața și spatele
    int spate, fata;
    // Coada circulară
    int dimensiune, *V;
    Coada(int s){
       fata = spate = -1; dimensiune = s; V = new int[s];
    }
    void Pozitionare(int valoare);
    int Decuplare();
    void AfisareCoada();
};
// Funcția de creare a cozii circulare
void Coada::Pozitionare(int valoare){
    if ((fata == 0 && spate == dimensiune-1) || (spate ==
(fata-1)%(dimensiune-1))){
        cout<<"\nCoada este plina!"; return;
    }
    else if (fata == -1){
        // Inserați primul element
        fata = spate = 0; V[spate] = valoare;
    }
    else if (spate == dimensiune-1 && fata != 0){
        spate = 0; V[spate] = valoare;
    }
    else {
        spate++; V[spate] = valoare;
    }
}
```

```
// Funcția de ștergere a elementului din coada circulară
int Coada::Decuplare(){
    if (fata == -1){
        cout<<"\nCoada este goala!"; return INT_MIN;
    }
    int info = V[fata]; V[fata] = -1;
    if (fata == spate){
        fata = -1; spate = -1;
    }
    else if (fata == dimensiune-1) fata = 0;
    else fata++;
    return info;
}
// Funcție care afișează elementele cozii circulare
void Coada::AfisareCoada(){
    if (fata == -1){
        cout<<"\nCoada este goala!"; return;
    }
    cout<<"\nElementele din coada circulara sunt: ";
    if (spate >= fata){
        for (int i = fata; i <= spate; i++)
            cout<<V[i]<<" ";
    }
    else {
        for (int i = fata; i < dimensiune; i++)
            cout<<V[i]<<" ";
        for (int i = 0; i <= spate; i++)
            cout<<V[i]<<" ";
    }
}
// Programul principal
int main(){
    Coada C(5);
    // Inserarea elementelor în coada circulară
    C.Pozitionare(14); C.Pozitionare(22); C.Pozitionare(13);
    C.Pozitionare(-6);
    // Afișarea elementelor din coada circulară
    C.AfisareCoada();
    // Ștergerea elementelor din coada circulară
    cout<<"\nAu fost sterse din coada elementele: ";
    cout<<C.Decuplare()<<" "<<C.Decuplare();
    // Afișarea elementelor ramase in coada circulară dupa
```

```
stergerea primelor doua
    C.AfisareCoada();
    // Inserarea elementelor în coada circulară
    cout<<"\nVor fi inserate in coada cateva elemente noi!";
    C.Pozitionare(9); C.Pozitionare(20); C.Pozitionare(5);
    // Afișarea elementelor din coada circulară dupa inserarea
altor trei elemente noi
    C.AfisareCoada(); C.Pozitionare(5);
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Elementele din coada circulara sunt: 10 11 12 13
Au fost sterse din coada elementele: 10 11
Elementele din coada circulara sunt: 12 13
Vor fi inserate in coada cateva elemente noi!
Elementele din coada circulara sunt: 12 13 14 15 16
Coada este plina!
```

```
Elementele din coada circulara sunt: 14 22 13 -6
Au fost sterse din coada elementele: 14 22
Elementele din coada circulara sunt: 13 -6
Vor fi inserate in coada cateva elemente noi!
Elementele din coada circulara sunt: 13 -6 9 20 5
Coada este plina!
```

Aplicații:

- *Managementul memoriei: locațiile de memorie neutilizate în cazul cozilor obișnuite pot fi utilizate în cozile circulare.*
- *Sistemul de trafic: în sistemul de trafic controlat de computer, cozile circulare sunt folosite pentru a aprinde semafoarele unul câte unul în mod repetat, conform orei setate.*
- *Programarea proceselor CPU: Sistemele de operare mențin adesea o coadă de procese care sunt gata de executare sau care așteaptă să apară un anumit eveniment.*

Problema 3 – Inversarea primelor K elemente ale cozii:

Fie avem un număr întreg k și o coadă de numere întregi. Trebuie să inversăm ordinea primelor k elemente ale cozii, lăsând celelalte elemente în aceeași ordine relativă. Numai următoarele operații standard sunt permise pentru coadă: adăugarea unui element x în spatele cozii, eliminarea unui element din fața cozii, returnarea numărului de elemente din coadă și găsirea elementului frontal.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
#include <stack>
#include <queue>
using namespace std;
// Funcție pentru a inversa primele K elemente ale cozii
void InversareK(
    int k, queue<int>& C){
    if (C.empty() == true || k > C.size()) return;
    if (k <= 0) return;
    stack<int> S;
    // Împingeți primele elemente K într-o stivă
    for (int i = 0; i < k; i++) {
        S.push(C.front()); C.pop();
    }
    // Stoarceți conținutul stivei din spatele cozii
    while (!S.empty()) {
        C.push(S.top()); S.pop();
    }
    // Eliminați elementele rămase și puneți-le în coadă la
sfârșitul cozii
    for (int i = 0; i < C.size() - k; i++) {
        C.push(C.front()); C.pop();
    }
}
// Funcția pentru a imprima coada la ecran
void Afisare(queue<int>& C){
    while (!C.empty()) {
        cout << C.front() << " "; C.pop();
    }
```

```
}
// Programul principal
int main(){
    queue<int> A;
    int k = 4; // Numărul de elemente pentru inversare
    // Introducem elementele cozii
    A.push(3); A.push(5); A.push(7); A.push(9);
    A.push(2); A.push(4); A.push(6); A.push(8);
    // Afișarea soluției după inversare
    cout<<"\nSolutia dupa inversarea primelor "<<k;
    cout<<" elemente este: \n\t";
    InversareK(k, A); Afisare(A);
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Solutia dupa inversarea primelor 4 elemente este:
        9 7 5 3 2 4 6 8
```

Observație:
Ideea este de a folosi o stivă auxiliară. Creăm o stivă goală. Efectuăm operațiile de adăugare a unui element x în spatele cozii și de eliminare a unui element din fața cozii. Scoatem k elementele din față și le fixăm unul câte unul în aceeași coadă.

Remarcă:

Complexitatea timpului:
- *O (n+k). Unde „n" este numărul total de elemente din coadă și „k" este numărul de elemente care trebuie inversate. Acest lucru se datorează faptului că, în primul rând, întreaga coadă este golită în stivă și, după aceea, primele elemente „k" sunt golite și extrase în același mod.*

Spațiu auxiliar:
- *O (n). Utilizarea stivei pentru stocarea valorilor în scopul inversării.*

Problema 4 – Interclasarea elementelor jumătăților unei cozi

Fie avem o coadă de numere întregi de lungime uniformă, rearanjați elementele intercalând prima jumătate a cozii cu a doua jumătate a cozii. Doar o stivă poate fi folosită ca spațiu auxiliar.

Prezentarea soluției problemei în limbajul C++

```
#include <iostream>
#include <stack>
#include <queue>
using namespace std;
// Funcție de intercalare a cozii
void intercalare(queue<int>& C){
    // Pentru a verifica numărul par de elemente
    if (C.size() % 2 != 0)
        cout << "Introduceti numarul par de numere intregi.\n";
    // Inițializați o stivă goală de tip int
    stack<int> S;
    int dim2 = C.size() / 2;
    // Împingeți prima jumătate de elemente în stivă
    for (int i = 0; i < dim2; i++) {
        S.push(C.front()); C.pop();
    }
    // Readuceți înapoi elementele stivei
    while (!S.empty()) {
        C.push(S.top()); S.pop();
    }
    // Extrageți elementele din prima jumătate a cozii și
puneți-le înapoi
    for (int i = 0; i < dim2; i++) {
        C.push(C.front()); C.pop();
    }
    // Împingeți din nou elementele din prima jumătate în stivă
    for (int i = 0; i < dim2; i++) {
        S.push(C.front()); C.pop();
    }
   // Intercalează elementele cozii și stivei
    while (!S.empty()) {
        C.push(S.top()); S.pop();
```

```
        C.push(C.front()); C.pop();
    }
}
// Funcția pentru a imprima coada la ecran
void Afisare(queue<int>& C){
    while (!C.empty()) {
        cout << C.front() << " "; C.pop();
    }
}
// Programul principal
int main(){
    queue<int> A;
    // Introducem elementele cozii
    A.push(1); A.push(3); A.push(5); A.push(7);
    A.push(2); A.push(4); A.push(6); A.push(8);
    // Afișarea soluției după interclasare
    cout<<"\nSolutia dupa interclasare celor doua jumatati ale
cozii: \n\t";
    intercalare(A); Afisare(A);
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Solutia dupa interclasare celor doua jumatati a cozii:
        1 2 3 4 5 6 7 8
```

```
Solutia dupa interclasare celor doua jumatati a cozii:
        8 7 6 5 4 3 2 1
```

```
Solutia dupa interclasare celor doua jumatati a cozii:
        18 27 16 25 14 23 12 21
```

Remarcă:

Complexitatea timpului: O (n).
Spațiu auxiliar: O (n).

Problema 5 – Cel mai mare multiplu al lui K:

Fie avem o coadă reprezentată ca un masiv unidimensional de numere întregi care nu sunt negative (conține elemente de la 0 la 9). Găsiți cel mai mare multiplu al numărului K ce poate fi format din câteva sau chiar toate elementele masivului. Același element poate apărea de mai multe ori în masiv, dar fiecare element din masiv poate fi utilizat o singură dată.

De exemplu, dacă avem un număr K=3 și masivul de intrare conține elementele: {5, 7, 5, 5}, atunci rezultatul la ieșire ar trebui să fie „5 5 5”. Suma cifrelor = 5 + 5 + 5 + 7 = 22, restul de la operația 22% 3 = 1. Deci eliminăm cea mai mică cifră care are restul '1'. Eliminăm 7% 3 = 1, deci cel mai mare număr divizibil cu 3 este: 555.

Prezentarea soluției problemei în limbajul C++:

```
#include <iostream>
// Acest antet descrie un set de algoritmi pentru a efectua anumite operații pe secvențe de valori numerice.
#include <numeric>
using namespace std;
// Numărul de cifre
#define dimV 10
#define K 5
// Funcția de sortare a vectorului de cifre folosind numărare
void SortareVector(int V[], int n){
    // Stocați numărul tuturor elementelor
    int count[dimV] = {0};
    for (int i = 0; i < n; i++)
        count[V[i]]++;
    // Depozităm
    int index = 0;
    for (int i = 0; i < dimV; i++)
        while (count[i] > 0)
           V[index++] = i, count[i]--;
}
// Eliminați elementele din vectorul V[] la indexurile ind1 și ind2
bool ElimiareAfisare(int V[], int n, int ind1, int ind2 = -1){
```

```
    for (int i = n-1; i >=0; i--)
        if (i != ind1 && i != ind2) cout << V[i] ;
}
// Returnează cel mai mare multiplu de K care poate fi format
folosind elementele vectorul V[]
bool MaxMultipluK(int V[], int n){
    // Suma tuturor elementelor vectorul V[]
    int suma = accumulate(V, V+n, 0);
    // Dacă suma este divizibilă cu K, nu este nevoie să
ștergeți un element
    if (suma%K == 0) return true ;
    // Sortați elementele vectorul V[] în ordine crescătoare
    SortareVector(V, n);
    // Găsim un semn distinctiv (rest)
    int  rest= suma % K;
    if (rest == 1){
        int rest1[2];
        rest1[0] = -1, rest1[1] = -1;
        // Elementele masivului transversal V[]
        for (int i = 0 ; i < n ; i++){
            if (V[i]%K == 1){
                ElimiareAfisare(V, n, i); return true;
            }
            if (V[i]%K == 2){
                if (rest1[0] == -1) rest1[0] = i;
                // Dacă a doua apariție
                else if (rest1[1] == -1) rest1[1] = i;
            }
        }
        if (rest1[0] != -1 && rest1[1] != -1){
            ElimiareAfisare(V, n, rest1[0], rest1[1]); return
true;
        }
    }
    else if (rest == 2){
        int rest2[2];
        rest2[0] = -1, rest2[1] = -1;
        // Traversează elemente vectorul V[]
        for (int i = 0; i < n; i++){
            if (V[i]%K == 2){
                ElimiareAfisare(V, n, i); return true;
            }
```

```
            if (V[i]%K == 1){
                if (rest2[0] == -1) rest2[0] = i;
                // Dacă a doua apariție
                else if (rest2[1] == -1) rest2[1] = i;
            }
        }
        if (rest2[0] != -1 && rest2[1] != -1){
            ElimiareAfisare(V, n, rest2[0], rest2[1]); return
true;
        }
    }
    cout << "Nu este posibil!"; return false;
}
// Programul principal
int main(){
    int V[] = {7, 5, 5, 5} ;
    int n = sizeof(V)/sizeof(V[0]);
    cout<<"Vectorul de elemente introdus este: ";
    for (int i=0;i<n;i++){
        cout<<V[i]<<" ";
    }
    cout<<"\nValoarea multiplului maxim al numarului "<<K<<"
este: ";
    MaxMultipluK(V, n); return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
Vectorul de elemente introdus este: 7 5 5 5
Valoarea multiplului maxim al numarului 5 este: 555
```

```
Vectorul de elemente introdus este: 7 5 3 1
Valoarea multiplului maxim al numarului 5 este: 753
```

Remarcă:

Complexitatea timpului: O (n).
Spațiu auxiliar: O (1).

Problema 6 – Automobile

Se consideră o coadă de automobile, la o stație de alimentare cu benzină, cu caracteristicile: numărul de înmatriculare (un câmp alfanumeric de 10 caractere), tipul automobilului (un câmp alfanumeric de 15 caractere), culoarea (un câmp alfanumeric de 10 caractere). Să se simuleze activitățile de alimentare cu benzină ale automobilelor din coadă: adăugarea unui automobil la coadă, eliminarea din capul cozii a automobilului alimentat si listarea tuturor automobilelor din coadă folosindu-se câte o funcție adecvat definită.

Rezolvare:

Evenimentele care pot avea loc cu această coadă sunt: adăugarea unui nou automobil la coadă (simulat prin apăsarea tastei a), eliminarea din coadă a automobilului alimentat (simulat prin apăsarea tastei e) și listarea automobilelor din coadă (simulat prin apăsarea tastei l).

Pentru un automobil sosit la coadă se vor cere: numărul de inmatriculare, tipul, culoarea și capacitatea rezervorului, iar la plecarea automobilului după ce a fost alimentat se vor afișa aceleași informații. Listarea automobilelor din coadă presupune afișarea acelorași informații pentru toate automobilele care sunt la coadă și urmează să se alimenteze cu benzină. Coadă automobilelor se va implementa ca un tablou de pointeri care se inițializează cu valoarea NULL.

Toate operațiile asupra unei cozi se fac pe la ambele capate ale listei și constau fie din adăugarea unei structuri de tip automobil la sfârșitul cozii fie din extragerea (ștergera) unei structuri de tip automobil la începutul cozii. Funcția qstore() va simula așezarea unui nou automobil la coadă și verifică dacă lista este completă. Funcția qretrieve() simulează eliberarea unui automobil din coadă și va returna NULL dacă nu mai sunt automobile în coadă și va returna de asemenea poziția următorului automobil care va fi eliberat, permițând în acest mod afișarea informațiilor despre automobilul ce va fi eliberat.

Prezentarea soluției problemei în limbajul C++

```
#include<iostream>
#include<string.h>
```

```
#include<stdlib.h>
#include<conio.h>
#define MAX 20
using namespace std;
/* Crearea structurii corespunzatoare*/
struct tipmasina{
    char numar[10],tip[15], culoare[10];
    int capacitate;
};
struct tipmasina *ptrmasina[MAX],*qretrive(void);
int freepos=0, recupos=0;
/* Descrierea functiei de adaugare automobil la coada, adau-
gare() */
void qstore(struct tipmasina *q){
    if(freepos==MAX){
        cout<<"\n Lista plina!";return;
    }
    ptrmasina[freepos]=q; freepos++;
}
/* Descrierea functiei de extragere automobil din coada */
struct tipmasina *qretrieve(void){
    if(recupos==freepos){
        cout<<"\n Nu mai sunt automobile la coada!";
        return NULL;
    }
    recupos++; return ptrmasina[recupos-1];
}
/* Descrierea functiei de adaugare automobil in coada */
void adaugare(){
    char r;
    struct tipmasina masina,*ptrmasina;
    do{
       cout<<"\n Introduceti in coada automobilul "<<freepos+1;
        r=getche();
         ptrmasina=(struct tipmasina*)malloc(sizeof(struct tip-
masina));
        if(!ptrmasina){
              cout<<"\n Memorie insuficienta pentru o noua alo-
care!"; return;
        }
        cout<<"\n Numarul de inmatriculare al automobilului: ";
        cin>>ptrmasina->numar;
```

```
        cout<<"\n Tipul automobilului: "; cin>>ptrmasina->tip;
        cout<<"\n Culoarea automobilului: ";
        cin>>ptrmasina->culoare;
        cout<<"\n Capacitatea rezervorului: ";
        cin>>ptrmasina→capacitate; qstore(ptrmasina);
    }
    while ((r!='d')&&(r!='D'));
}
/* Descrierea functiei de extragere (stergere) din stiva */
void stergere(){
    struct tipmasina *ptrmasina;
    if((ptrmasina=qretrieve())==NULL) return;
    cout<<"\n Date despre automobilul iesit de la coada:";
    cout<<"\n ================================================";
     cout<<"\n Automobilul "<<ptrmasina->numar<<" "<<ptrmasina-
>tip<<" "<<ptrmasina->culoare<<" "<<ptrmasina->capacitate;
    cout<<"\n ================================================";
    getch();
}
/* Descrierea functiei de listare a tuturor automobilelor */
void listare(){
    int t;
    cout<<"\n Numar \tTip \tCuloare \tCapacitate(l)";
    cout<<"\n ================================================";
    for(t=recupos;t<freepos;++t)
         cout<<"\n   "<<ptrmasina[t]->numar<<"\t"<<ptrmasina[t]-
>tip<<"\t"<<ptrmasina[t]->culoare<<"\t\t"<<ptrmasina[t]->capac-
itate;
    cout<<"\n ================================================";
    getch();
}
/* PROGRAMUL PRINCIPAL */
int main(){
    int t; char rasp;
    for(t=0;t<MAX;++t) ptrmasina[t]=NULL;
    for(;;){
        system("CLS");
   cout<<"\n a - Asezare automobilului la coada pentru ali-
mentare";
     cout<<"\n e - Alimentare si iesire automobil de la coada";
     cout<<"\n l - Consultare si listare automobil din coada";
     cout<<"\n t - Terminare program";
```

```
     cout<<"\n\n Alegeti operatiunea dorita:"; rasp=getche();
/* Descrierea MENIULUI */
        switch (rasp){
             case 'a': cout<<"\n Soseste si se introduce un nou
automobil la coada";
            adaugare(); break;
              case 'e': cout<<"\n Se alimenteaza si iese primul
automobil de la coada";
            stergere(); break;
              case 'l': cout<<"\n Se listeaza automobilele care
se gasesc la coada";
            listare(); break;
             case 't': cout<<"\n Terminarea gestionarii cozii de
automobile";
            exit(0);
        }
    }
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
a - Asezare automobilului la coada pentru alimentare
e - Alimentare si iesire automobil de la coada
l - Consultare si listare automobil din coada
t - Terminare program

Alegeti operatiunea dorita:
```

Alegem operațiunea a, apoi introducem datele a 4 automobile, după aceasta vom afișa lista automobilelor din coadă (introducem litera l):

```
Se listeaza automobilele care se gasesc la coada
Numar  Tip     Culoare         Capacitate(l)
=============================================
123     SUV     Brown          2300
111     DIE     Red            6000
654     DEX     White          5000
765     SUV     Yellow         2300
=============================================
```

6.5 Modele de probleme propuse

1. Elaborați subprograme în C++ pentru următoarele probleme simple.

- ***a.*** *Să se implementeze o stivă folosind două cozi.*
- ***b.*** *Să se implementeze o coadă folosind două stive. (Se vor utiliza apeluri recursive ale unor funcţii ce se contorizează ca folosirea unei stive).*
- ***c.*** *Să se implementeze o stivă cu valori întregi şi o funcţie care obţine valoarea maximă din stivă. Se cere ca funcţia să aibă complexitate de timp constantă ⇒ O(1).*

2. Elaborați un program în C++ pentru următoarea problemă cu fișiere.

- ***a.*** *Într-o structură de date de tip coadă au fost adăugate, în această ordine, următoarele valori: 3, 10, 2, 8 şi 6 (datele sunt citite din fișier). Care este ultima valoare care s-a extras din coadă dacă s-au efectuat, în această ordine, următoarele operaţii: extragerea unui element, adăugarea valorii 100, extragerea a trei elemente.* ***(Răspuns corect: 8)***
- ***b.*** *Se consideră o stivă, iniţial vidă, în care s-au introdus în ordine valorile 1, 2, 3 şi o coadă, iniţial vidă, în care au fost introduse, în ordine, valorile 6, 5, 4 (datele sunt citite din fișier pentru ambele structuri). Care va fi valoarea elementului din vârful stivei dacă se extrag toate elementele din coadă şi se adaugă, în ordinea extragerii, în stiva dată?* ***(Răspuns corect: 4)***
- ***c.*** *Se consideră o coadă, în care au fost introduse iniţial, în această ordine, două numere: 2 şi 1. Notăm cu AD X operaţia prin care se adaugă informaţia X în coadă şi cu EL operaţia prin care se elimină un element din coadă. Asupra cozii se efectuează, exact în această ordine, operaţiile AD 10; AD 15; EL; AD 4; EL; AD 20; EL. Rezultatul se va păstra într-un fișier. Care este conţinutul cozii după executarea operaţiilor de mai sus?* ***(Răspuns corect: 15 4 20)***

3. Elaborați un program în C++ pentru următoarele probleme.

3.1 *Se dă un vector cu n întregi și un număr k. Aflați valoarea maximă (valoarea minimă sau valoarea medie) pentru fiecare grupare de k numere de pe poziții consecutive.*

3.2 *Se dă un vector cu datele pentru n clienţi la un server. Pentru fiecare client, datele cunoscute sunt ora la care se conectează şi ora la care se deconectează. Aflaţi numărul maxim de clienţi conectaţi în acelaşi timp la server. Se cere complexitate de timp O(n).*

3.3 *Într-o coadă ale cărei elemente reţin informaţii numere întregi, au fost introduse, în această ordine, numerele 6,5,4,3,2,1. Asupra cozii se efectuează, în această ordine, următoarele operaţii: se elimină un element, se adaugă două elemente cu valorile 6 şi respectiv 7 şi apoi se elimină 3 elemente. Care sunt ultimele 3 valori eliminate?*

3.4 *Se consideră o coadă, în care au fost introduse inițial, în această ordine, două numere 2 şi 1. Notăm cu AD X operaţia prin care se adaugă informaţia X în coadă şi cu EL operaţia prin care se elimină un element din coadă. Asupra cozii se efectuează, exact în această ordine, operaţiile AD 5; EL; AD 4; EL; EL; AD 8; AD 9; EL. Care este conţinutul cozii după executarea operaţiilor de mai sus?*

3.5 *O coadă este o modalitate folositoare de a stoca date care provin în mod asincronic de la un microcontroler periferic, dar care nu pot fi citite imediat. Un bun exemplu ar fi stocarea de biți proveniți de la un UART (Universal asynchronous receiver/transmitter). Un buffer FIFO stochează date pe principiul „primul venit - primul servit“. Structura de stocare este un spațiu alăturat de memorie. Datele sunt scrise în capul cozii și citite de la coadă. Dacă parcurgerea are loc de la coadă spre cap, buffer-ul este gol. Dar dacă parcurgerea este de la cap spre coadă, implementarea trebuie să definească dacă cea mai veche dată trebuie scoasă sau dacă scrierea nu s-a terminat.*

6.6 Portofoliul elevului pentru coadă

Se consideră o coadă ce memorează valori întregi pozitive. Elaborați subprograme pentru următoarele operațiuni:

a. Introduceți un număr nou în coadă;
b. Afișați elementele pare și impare ale cozii;
c. Afișați elementele pare pozitive și pare negative ale cozii;
d. Afișați elementele impare pozitive și impare negative ale cozii;
e. Afișați elementele în ordine crescătoare ale cozii;
f. Afișați elementele în ordine descrescătoare ale cozii;
g. Să se șteargă primul nod (ultimul nod)care conține valoarea x.
h. Să se șteargă nodurile care conțin pătrate perfecte.
i. Să se șteargă nodurile care conțin numere Fibonacci.

Fiecare elev din subgrupă va realiza o bibliotecă ce va include operațiile de mai sus, cu condiția că elementele inițiale ale cozii sunt citite din fișier, fiecare elev va avea o coadă individuală conform numărului de ordine.

Nr.	Secvența de numere	Nr.	Secvența de numere
1	50 31 30 35 82 85 16 18 58 61	**11**	21 37 56 32 44 71 99 87 72 90
2	79 59 45 14 57 23 73 83 51 62	**12**	19 12 49 18 37 71 39 68 42 98
3	77 81 66 84 95 43 38 65 15 76	**13**	75 46 97 40 57 70 54 73 14 59
4	26 64 39 13 17 10 40 55 29 48	**14**	56 31 44 27 96 80 81 86 58 65
5	47 74 27 91 75 88 33 25 93 42	**15**	53 21 50 67 83 69 93 32 43 17
6	67 22 69 96 46 97 28 12 41 36	**16**	41 95 89 77 29 33 87 51 99 72
7	94 63 92 34 53 78 11 52 19 86	**17**	61 76 45 24 79 85 52 92 15 36
8	70 24 89 80 60 49 68 20 98 54	**18**	82 64 74 38 23 16 63 30 10 62
9	90 60 11 78 35 84 22 88 20 34	**19**	62 12 38 79 43 90 17 23 80 92
10	55 48 28 26 66 25 91 94 13 47	**20**	30 42 50 49 41 11 71 58 83 44

6.7 Model de test grilă pentru coadă

Nr.	Conținutul	Punctaj
1	*Cozile şi stivele sunt structuri de date logice şi omogene.* ◯ adevărat ◯ fals	*2p*
2	*Coada se poate implementa folosind un vector circular de dimensiune n.* ◯ adevărat ◯ fals	*2p*
3	*Funcția realloc() din C++ alocă un bloc de memorie neinițializată și returnează un pointer gol (void) la primul octet al blocului de memorie alocat dacă alocarea reușește.* ◯ adevărat ◯ fals	*2p*
4	*Care sunt aplicațiile de bază ale cozii?* ☐ Implementare în planificarea proceselor ☐ Implementare în spooling de tipărire ☐ Implementare în cautarea în lățime ☐ Se aplică doar pentru primele două	*6p*
5	*Coada este o structură de date liniara ce lucrează cu o colecție de date având câteva operații principale:* ☐ isFull() ☐ EnQueue ☐ peek() ☐ răspunsul corect lipsește	*6p*
6	*Care sunt modalitățile de implementare a structurii dinamice coada?* ☐ array ☐ folosirea containerului queue ☐ LLSÎ ☐ LCSÎ & LCDÎ	*6p*
7	*Care nu este operație de bază a structurii dinamice coada?* ◯ emplace() ◯ front() ◯ back() ◯ push()	*2p*
8	*Care este denumirea funcției din C++ ce returnează valoarea elementului din listă ce a fost adăugat la urmă?* ◯ push() ◯ pop() ◯ back() ◯ push()	*2p*

9	*Care este denumirea funcției din C++ ce permite schimbarea conținutului a două cozi, unde cozile trebuie să fie de același tip, deși dimensiunile pot diferi?* ◯ realloc() ◯ calloc() ◯ malloc() ◯ swap()	*2p*
10	*Care este denumirea funcției din C++ STL ce elimină un element din coadă, adică un element este îndepărtat sau scos din coadă întotdeauna din partea din față a cozii?* ◯ DeQueue ◯ isQueue ◯ EnQueue ◯ notQueue	*2p*
11	*Se consideră o coadă în care inițial au fost introduse, în această ordine, elementele 1 şi 2. Dacă se notează cu AD(x) operația prin care se adaugă informația x în coadă, şi cu EL operația prin care se elimină un element din coadă, care este rezultatul executării secvenței EL; AD(3); EL; AD(4); AD(5); ?* ◯ 1 4 5 ◯ 3 4 5 ◯ 5 4 2 ◯ 5 4 3	*2p*
12	*Se consideră o coadă C inițial vidă. În coadă se introduc în această ordine elementele: 3, 5, 6, 7, 10, 13. Se fac apoi următoarele operații: se elimină un element din coadă, apoi se adaugă elementul cu valoarea 8, se elimină apoi două elemente din coadă. Care va fi ultimul element al cozii?* ◯ 3 ◯ 13 ◯ 7 ◯ 8	*2p*

Barem de notare propus

Nota	10	9	8	7	6	5	4	3	2
%	100-95	94-88	87-78	77-63	62-48	47-33	32-21	20-10	9-5
Punctaj	36-34	35-32	31-28	27-23	22-17	16-12	11-8	7-4	3-2

7. STRUCTURI ARBORESCENTE

7.1 Introducere

Organizarea liniară de tip listă este adecvată pentru aplicațiile în care datele (elementele din listă) formează o mulțime omogenă și deci se află pe același nivel. În multe aplicații, este strict necesară organizarea ierarhică pentru studiul și rezolvarea problemelor. Arborii sunt un caz particular al grafurilor. Aceștia sunt compuși dintr-o serie de noduri interconectate în care se găsesc informații.

Definiția 1: *Arborele este un graf neorientat conex fără cicluri, în care unul din noduri este desemnat ca rădăcină. Nodurile pot fi așezate pe niveluri începând cu rădăcina care este plasată pe nivelul 1. Rădăcina unui arbore este un nod special care ajută la delimitarea arborelui pe nivele. Acest nod se află pe cel mai înalt nivel din arbore.*

Arborele devine mai ușor de parcurs, deoarece plasarea nodurilor pe nivele duce la o structurare mai eficientă a informațiilor. Algoritmii ce folosesc arbori sunt foarte ușor de implementat recursiv, deoarece la fiecare pas, putem separa arborele în mai mulți arbori mai mici.

Definiția 2: *Un arbore binar este o mulțime de noduri care îndeplinesc următoarele condiții: fiecare nod are 0, 1 sau 2 succesori; fiecare nod are un singur predecesor, cu excepția rădăcinii care nu are niciunul; succesorii fiecărui nod sunt ordonați (fiul stâng, fiul drept; dacă este unul singur trebuie să-l menționăm).*

Definiția 3 (definiția recursivă):

- *(Baza:) Arborele fără niciun nod este un arbore binar.*
- *(Pasul recursiv:) Fie a și b doi arbori binari, iar n un nod. Atunci arborele care îl are pe n ca rădăcină, pe a ca subarbore stâng și pe b ca subarbore drept este un arbore binar.*

7.2 Structura unui arbore

Un arbore poate fi reprezentat sub următoarea formă:

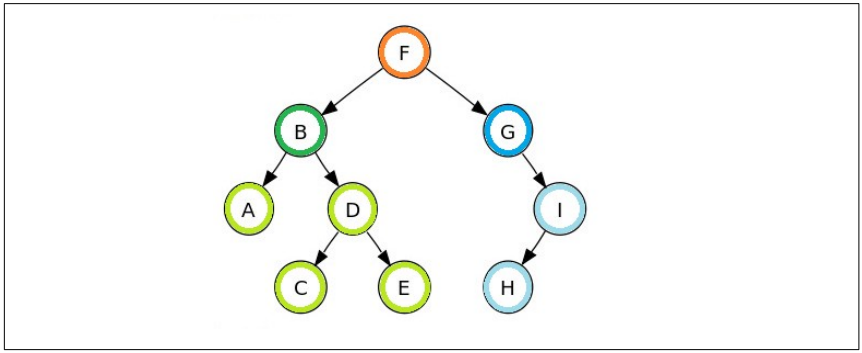


Figura 6.1 – Reprezentarea grafică a unui arbore

Arborele de mai sus are ca rădăcină nodul F, de asemenea putem privi nodul B și G ca rădăcini a doi noi arbori mai mici (sub-arbori). Observăm că arborele reprezentat are două brațe: brațul stâng B și brațul drept G.

În continuare vom prezenta câteva noțiuni utile în înțelegerea arborilor. Vom folosi ca exemplu arborele de mai sus cu rădăcina F. Arborele poate fi împărțit în nivele astfel :

- *F(nodul rădăcină) – Nivelul 1*
- *B și G – Nivelul 2*
- *A, D și I – Nivelul 3*
- *C, E și H – Nivelul 4*

Unele noțiuni fundamentale le vom explica în tabelul de mai jos:

Noțiuni utile	***Exemplu***
Un nod A este descendent al unui alt nod B, dacă este situat pe un nivel mai mare decât B şi există un lanţ care le uneşte şi nu trece prin rădăcină.	*În arborele de mai sus E este descendentul lui B și al lui A.*

Noțiuni utile	*Exemplu*
Un nod A este fiu/descendent direct al unui alt nod B, dacă este situat pe nivelul imediat următor nodului B și există muchie între A și B.	*În arborele de mai sus, B și G sunt fii lui F(nodul rădăcină), A este fiul lui B, H este fiul lui I, ș.a.m.d.*
Un nod A este ascendent al unui alt nod B, dacă este situat pe un nivel mai mic decât B și există lanț care le unește și nu trece prin rădăcină.	*În arborele de mai sus, G este ascendentul lui H, B este ascendentul lui C.*
Un nod A este părinte al unui alt nod B, dacă este situat pe nivelul imediat superior nodului B și există muchie între A și B.	*În arborele de mai sus, F este părintele lui B, D este părintele lui E, I este părintele lui H, ș.a.m.d..*
Două noduri sunt frați dacă au același părinte.	*În arborele de mai sus, B și G sunt frați, A și D sunt frați.*
Un nod este frunză dacă nu are nici un fiu, adică se află pe ultimul nivel.	*În arborele nostru frunzele sunt C, E și H.*

Remarcă:

- *Arborii sunt structuri de date dinamice şi omogene. Cele mai comune utilizări ale arborilor sunt căutarea în volume mari de date şi reprezentarea de structuri organizate ierarhic.*
- *Orice arbore binar are doi subarbori: stâng si drept. Arborii binari pot fi arbori de căutare și arbori de selecţie; ei se caracterizează prin faptul că fiecare nod are o"cheie" reprezentată printr-o informație specifică de identificare a nodului. Cheia permite alegerea unuia din cei doi subarbori în funcţie de o decizie tip "mai mic", "mai mare", etc.*
- *Un arbore este echilibrat dacă pentru orice subarbore diferența dintre înalțimile subarborilor săi este cel mult 1.*
- *Doi subarbori pot fi în relație de incluziune, când un subarbore este inclus în celălalt sau de excluziune, când nu au noduri comune.*

7.3 Operații principale efectuate

7.3.1 Arbori oarecare

Pentru implementarea unui arbore oarecare în C++ se va folosi o structură de forma:

```
typedef double TipArbore;
// structura care reprezinta un nod din arbore
struct NodArbore{
     TipArbore Informatii; //informatiile utile stocate in nod
     int numarLegaturi; //numarul de legaturi catre fii
     NodArbore **Legaturi; //vector de legaturi catre fii
};
```

Fiecare nod conţine informaţiile utile, un întreg care reţine numărul de fii şi un vector de pointeri către fii. Principalele operaţii care pot fi implementate pe un arbore oarecare sunt:

Operația	Descrierea
Parcurgere arbore	*Presupune obţinerea unei liste ce conţine toate informaţiile utile din arbore.*
Adăugare nod	*Adaugă un nod în arbore după un anumit criteriu (de exemplu la un anumit nod părinte). Operaţia presupune alocarea memoriei pentru nod, copierea informaţiei utile şi modificarea legăturii părintelui.*
Ștergere nod	*Presupune dealocarea memoriei pentru nodul respectiv şi pentru toţi descendenţii săi, şi modificarea legăturii părintelui.*
Căutare element	*Presupune obţinerea unui pointer la un nod pe baza unui criteriu de regăsire.*

7.3.2 Arbori binari

Pentru implementarea unui arbore binar în C++ se va folosi o structură de forma:

```
typedef double TipArbore;
// structura care reprezinta un nod din arbore binar
struct NodArbore{
   TipArbore Informatii; //informatiile utile stocate in nod
   NodArbore *Stanga, *Dreapta; //vector de legaturi catre fii
};
```

Operațiile de parcurgere a arborilor binari se simplifică mult dacă vom considera că fiecare nod al arborelui subordonează un subarbore binar stâng și un subarbore binar drept. Având în vedere această observație se vor defini subprograme recursive care utilizează tehnica Divide et Impera (această tehnică de programare de va studia mai detaliat mai târziu).

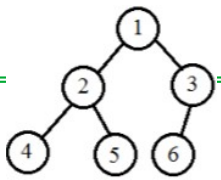
Parcurgerile arborilor binari sunt cele mai frecvente operații utilizate pe arbori. Parcurgerea unui arbore înseamnă vizitarea fiecărui nod al arborelui o singură dată, cu scopul prelucrării informației memorate în acel nod.

Dintre cele mai utilizate parcurgeri sunt parcurgerile în adâncime (algoritmul DFS) și pe niveluri (algoritmul BFS). Principalele parcurgeri în adâncime ale unui arbore binar sunt: inordine, preordine și postordine.

Metode specifice arborilor binari :

- *Parcurgerea in inordine (stânga –vârf – dreapta SVD) – se parcurge mai întâi subarborele stâng, apoi vârful, apoi subarborele drept.*
- *Parcurgerea in preordine (vârf- stânga – dreapta VSD) – se parcurge mai întâi vârful, apoi subarborele stâng, apoi subarborele drept.*
- *Parcurgerea în postordine (stânga – dreapta – vârf SDV) – se parcurge mai întâi subarborele stâng, apoi subarborele drept și la sfârșit vârful.*

Pentru a înțelege mai bine cele 3 tipuri de parcurgeri, vom analiza un exemplu simplu. Fie că avem un arbore cu 6 noduri. Se cere să se stabilească parcurgerea acestor noduri în: preordine, inordine și postordine.



Tabelul reprezentativ:

n	1	2	3	4	5	6
S	2	4	6	0	0	0
D	3	5	0	0	0	0

- *Parcurgerea în preordine, RSD:* **1 2 4 5 3 6.**
- *Parcurgerea în inordine, SRD:* **4 2 5 1 6 3.**
- *Parcurgerea în postordine, SDR:* **4 5 2 6 3 1.**

Parcurgerea pe niveluri – se vizitează rădăcina, apoi toţi fiii nodului rădăcină, de la stânga spre dreapta şi se continuă în acest mod pe toate nivelurile. Pentru arborele din figura recentă, şirul parcurgerii pe niveluri este: **1 2 3 4 5 6.**

Vom prezenta în continuare un mic exemplu de program care utilizează operaţiile de bază cu tipul de date dinamice, arbori binari.

```
#include<iostream>
#include<stdlib.h>
#include<conio.h>
using namespace std;
struct nod{
    int Informatii;
    nod *Dreapta, *Stanga;
};
nod *v, *aux;
int n, opt;
//Inserare nod în arborele binar
void inserare(nod *&c, int n){
    if(c)
    if(c->Informatii==n)
    cout<<"Nod deja inserat. Inserati altul!";
    else if(c->Informatii<n)
    inserare(c->Stanga, n);
    else inserare(c->Dreapta, n);
    else{
        c=new nod; c->Dreapta=c->Stanga=0; c->Informatii=n;
    }
```

```
}
//Parcurge arborele binar
void parcurg(nod *c){
    if(c){
        parcurg(c->Dreapta); cout<<c->Informatii<<" ";
        parcurg(c->Stanga);
    }
}
//In cazul in care un nod care urmeaza a fi sters, subordoneaza
doi arbori, se apeleaza functia _supl
void functie_supl(nod *&c, nod *&f){
    if(f->Stanga) functie_supl(c, f->Stanga);
    else {
        c->Informatii=f->Informatii; aux=f; f=f->Dreapta;
        delete aux;
    }
}
//Sterge un nod din arborele binar
void sterg(nod *&c, int n){
    nod *f;
    if(c)
    if(c->Informatii==n)
    if(c->Dreapta==0 && c->Stanga==0){
        delete c;
        c=0;
    }
    else if(c->Dreapta==0){
        f=c->Stanga;
        delete c;
        c=f;
    }
    else if(c->Stanga==0){
        f=c->Dreapta;
        delete c;
        c=f;
    }
    else
    functie_supl(c, c->Dreapta);
    else
    if(c->Informatii<n) sterg(c->Stanga, n);
    else sterg(c->Dreapta, n);
    else cout<<"Eroare";
```

```
}
// Programul principal
int main(){
    v=0;
    // Meniul de optiuni
    do{
        cout<<"\n1 - Inserare nod.";
        cout<<"\n2 - Parcurgere arbore.";
        cout<<"\n3 - Stergere nod.";
        cout<<"\nAlegeti o optiune:"; cin>>opt;
        system("CLS");
        switch(opt){
            case 1: cout<<"Introduceti nodul: "; cin>>n;
            inserare(v, n); break;
            case 2: parcurg(v); break;
            case 3: cout<<"Introduceti nodul pentru sters: ";
            cin>>n;
            sterg(v, n); break;
        }
    }
    while(opt!=4);
    getch();
}
```

Rezultatul obținut în urma execuției secvenței de cod C++ :

```
1 - Inserare nod.
2 - Parcurgere arbore.
3 - Stergere nod.
Alegeti o optiune:
```

Alegem operațiunea 1, apoi introducem 4 noduri, după aceasta vom afișa parcurgerea arborelui binar:

```
1 - Inserare nod.
2 - Parcurgere arbore.
3 - Stergere nod.
Alegeti o optiune:2
1 3 5 9
```

7.3.3 Arbori binari de căutare

Arborii binari de căutare sunt arbori binari în care nodurile sunt memorate ordonat în funcţie de o cheie. Toate nodurile din arbore au în subarborele stâng noduri care au chei mai mici şi în subarborele drept chei mai mari.

Arborii de căutare permit regăsirea rapidă a informaţiilor (O(log2 n)) atât timp cât arborele este echilibrat. În cazul cel mai defavorabil, timpul de căutare este identic cu cel al unei liste simplu înlănţuite.

Pentru implementarea unui arbore binar de căutare în C++ se va folosi o structură de forma:

```
typedef double TipArbore;
// structura care reprezinta un nod din arbore binar de cautare
struct NodArbore{
   TipArbore Informatii; //informatiile utile stocate in nod
   NodArbore *Stanga, *Dreapta; //vector de legaturi catre fii
   NodArbore(TipArbore informatii, NodArbore *stanga = NULL,
NodArbore *dreapta = NULL): Informatii(informatii),
Stanga(stanga), Dreapta(dreapta){}
};
```

Principalele operaţii care pot fi implementate pe un arbore binar de căutare sunt:

Operația	Descrierea
Adăugare nod	*Se caută, folosind algoritmul de căutare, poziţia în arbore, se alocă memoria şi se face legătura cu nodul părinte.*
Ștergere nod	*Se caută nodul de şters şi se şterge nodul. Subarborele drept este avansat în locul nodului şters, iar subarborele stâng este mutat ca fiu al celui mai mic element din subarborele drept.*
Căutare element	*Se compară cheia cu nodul curent. Dacă este egală, am găsit nodul, dacă este mai mică căutăm în subarborele stâng, altfel căutăm în subarborele drept. Căutarea se opreşte când nodul a fost găsit sau s-a atins baza arborelui.*

Implementarea operațiilor de bază pe un arbore binar de căutare este prezentată în următoarea bibliotecă:

```
#ifndef ARBORE_H_INCLUDED
#define ARBORE_H_INCLUDED
typedef double TipArbore;
// un nod din arbore
struct NodArbore{
    // informatia utila
    TipArbore Date;
    // legaturile catre subarbori
    NodArbore *Stanga, *Dreapta;
    // constructor pentru initializarea unui nod nou
    NodArbore(TipArbore date,NodArbore *stanga = NULL, NodAr-
bore *dreapta = NULL):Date(date), Stanga(stanga),
Dreapta(dreapta){}
};
// Arborele este manipulat sub forma unui pointer catre
radacina
typedef NodArbore* Arbore;
// Creaza un arbore vid
Arbore ArbCreare(){
    return NULL;
}
// Testeaza daca un arbore este vid
bool ArbEGol(Arbore& arbore){
    return arbore == NULL;
}
// Adauga un element intr-un arbore de cautare
void ArbAdauga(Arbore& arbore, TipArbore date){
    // Cazul 1: arbore vid
    if (ArbEGol(arbore)){
        arbore = new NodArbore(date); return;
    }
    // Cazul 2: arbore nevid
    if (date < arbore->Date)
    // daca exista subarborele stang
        if (arbore->Stanga != NULL)
        // inseram in subarbore
        ArbAdauga(arbore->Stanga, date);
        else
        // cream subarborele stang
```

```
        arbore->Stanga = new NodArbore(date);
        if (date > arbore->Date)
        // daca exista subarborele drept
        if (arbore->Dreapta != NULL)
        // inseram in subarbore
        ArbAdauga(arbore->Dreapta, date);
        else
        // cream subarborele drept
        arbore->Dreapta = new NodArbore(date);
}
// Functie privata de stergere a unui nod
void __ArbStergeNod(Arbore& legParinte){
    // salvam un pointer la nodul de sters
    Arbore nod = legParinte;
    // daca avem un subarbore drept
    if (nod->Dreapta != NULL){
        // facem legatura
        legParinte = nod->Dreapta;
        // daca avem si un subarbore stang
        if (nod->Stanga){
            // cautam cel mai mic element din subarborele drept
            Arbore temp = nod->Dreapta;
            while (temp->Stanga != NULL)
            temp = temp->Stanga;
            // si adaugam subarborele stang
            temp->Stanga = nod->Stanga;
        }
    }
    else
    // daca avem doar un subarbore stang
    if (nod->Stanga != NULL)
    // facem legatura la acesta
    legParinte = nod->Stanga;
    else
    // daca nu avem nici un subnod
    legParinte = NULL;
    // stergem nodul
    delete nod;
}
// Sterge un nod dintr-un arbore de cautare
void ArbSterge(Arbore& arbore, TipArbore date){
    // Cazul 1: arbore vid
```

```
    if (ArbEGol(arbore)) return;
    // Cazul 2: stergere radacina
    if (arbore->Date == date){
    // salvam un pointer la radacina
    Arbore nod = arbore;
    // daca avem un subarbore drept
    if (nod->Dreapta){
        // facem legatura
        arbore = nod->Dreapta;
        // daca avem si un subarbore stang
        if (nod->Stanga){
            // cautam cel mai mic element din subarborele drept
            Arbore temp = nod->Dreapta;
            while (temp->Stanga != NULL)
            temp = temp->Stanga;
            // si adaugam subarborele stang
            temp->Stanga = nod->Stanga;
        }
    }
    else
    // daca avem doar un subarbore stang
    if (nod->Stanga != NULL)
    // facem legatura la acesta
    arbore = nod->Stanga;
    else
    // daca nu avem nici un subnod
    arbore = NULL;
    // stergem vechea radacina
    delete nod;
    return;
    }
    // Cazul 3: stergere nod in arbore nevid cautam legatura la
nod in arbore si stergem nodul (daca exista)
    Arbore nodCurent = arbore;
    while (true){
        if (date < nodCurent->Date)
        if (nodCurent->Stanga == NULL)
        break; // nodul nu exista
        else if (nodCurent->Stanga->Date == date)
        // nodul de sters este descendentul stang
        __ArbStergeNod(nodCurent->Stanga);
        else
```

```
        // continuam cautarea in subarborele stang
        nodCurent = nodCurent->Stanga;
        else if (nodCurent->Dreapta == NULL)
        break; // nodul nu exista
        else if (nodCurent->Dreapta->Date == date)
        // nodul de sters este descendentul drept
        __ArbStergeNod(nodCurent->Dreapta);
        else
        // continuam cautarea in subarborele stang
        nodCurent = nodCurent->Dreapta;
    }
}
// Cauta recursiv un nod in arborele de cautare
bool Cautare(Arbore& arbore, TipArbore info){
    // conditia de oprire din recursie
    if (arbore == NULL) return false;
    // verificam daca am gasit nodul
    if (arbore->Date == info) return true;
    // daca cheia este mai mica
    if (arbore->Date < info)
    // cautam in subarborele stang
    return Cautare(arbore->Stanga, info);
    else
    // altfel cautam in subarborele drept
    return Cautare(arbore->Dreapta, info);
}

#endif // ARBORE_H_INCLUDED
```

Remarcă:

- *Elaborați un program în limbajul C++ care va aplica operațiile realizate în acest fișier antet.*
- *De asemenea, la dorință, se poate de îmbunătățit acest fișier antet cu alte subprograme pe care le veți aplica la rezolvarea problemelor.*

7.4 Modele de probleme rezolvate

Problema 1 – Operații uzuale cu arbori

Să se realizeze un program ce permite implementarea unui arbore binar de căutare, precum și a operațiilor uzuale cu acesta. Programul permite afișarea pe ecran a unui meniu cu următoarele operații posibile:

[1] Citirea unei valori de la tastatură și inserarea acesteia în arbore;
[2] Afișarea arborelui în preordine;
[3] Afișarea arborelui în inordine;
[4] Afișarea arborelui în postordine;
[5] Ștergerea unui subarbore, descendent dintr-un nod specificat;
[6] Căutarea unui nod în arbore;
[0] Ieșire din program.

Prezentarea soluției problemei în limbajul C++

```
#include<conio.h>
#include <iostream>
using namespace std;
/* definire structura arbore */
struct NOD{
    int x;
    struct NOD *NOD_stanga; struct NOD *NOD_dreapta;
};
/* functie creare nod nou */
struct NOD *creare_nod(int x){
    struct NOD *nod_nou;
    /* alocare memorie nod*/
    nod_nou=(struct NOD *)malloc(sizeof(struct NOD));
    if (nod_nou==NULL){
        cout<<"Eroare: Memoria nu a putut fi alocata! \n";
        return NULL;
    }
    /* initializare informatii */
    nod_nou->x=x;
    nod_nou→NOD_stanga=NULL; nod_nou->NOD_dreapta=NULL;
    return nod_nou;
```

```
}
/* inserare nod in arbore */
struct NOD *inserare_nod(struct NOD *prim, int x){
    struct NOD *nod_nou, *nod_curent, *nod_parinte;
    nod_nou=creare_nod(x);
    if (prim==NULL){
        /* arborele este vid */
        prim=nod_nou;
        cout<<"A fost adaugat primul nod: "<<prim->x<<"\n";
        return prim;
    }
    else{
        /* pozitionare in arbore pe parintele nodului nou */
        nod_curent=prim;
        while (nod_curent!=NULL){
            nod_parinte=nod_curent;
            if (x<nod_curent->x) /* parcurgere spre stanga */
            nod_curent=nod_curent->NOD_stanga;
            else /* parcurgere spre dreapta */
            nod_curent=nod_curent->NOD_dreapta;
        }
        /* creare legatura nod parinte cu nodul nou */
        if (x<nod_parinte->x){
        /* se insereaza la stanga nodului parinte */
            nod_parinte->NOD_stanga=nod_nou;
             cout<<"Nodul "<<x<<" a fost inserat la stanga nodu-
lui "<<nod_parinte->x<<"\n";
        }
        else{
            /* se insereaza la dreapta nodului parinte */
            nod_parinte->NOD_dreapta=nod_nou;
                cout<<"Nodul "<<x<<" a fost inserat la dreapta
nodului "<<nod_parinte->x<<"\n";
        }
        return prim;
    }
}
/* parcurgere arbore in preordine */
void afisare_preordine(struct NOD *prim){
    if (prim!=NULL){
        /* parcurgere radacina, stanga, dreapta */
        cout<<prim->x<<"\n";
```

```
        afisare_preordine(prim->NOD_stanga);
        afisare_preordine(prim->NOD_dreapta);
    }
}
/* parcurgere arbore in inordine */
void afisare_inordine(struct NOD *prim){
    if (prim!=NULL){
        /* parcurgere stanga, radacina, dreapta */
        afisare_inordine(prim->NOD_stanga);
        cout<<prim->x<<"\n";
        afisare_inordine(prim->NOD_dreapta);
    }
}
/* parcurgere arbore in postordine */
void afisare_postordine(struct NOD *prim){
    if (prim!=NULL){
        /* parcurgere stanga, dreapta, radacina */
        afisare_postordine(prim->NOD_stanga);
        afisare_postordine(prim->NOD_dreapta);
        cout<<prim->x<<"\n";
    }
}
/* stergerea unui arbore sau subarbore */
struct NOD *stergere_arbore(struct NOD *tmp){
    if (tmp!=NULL){
        stergere_arbore(tmp->NOD_stanga);
        stergere_arbore(tmp→NOD_dreapta); free(tmp);
    }
    return NULL;
}
/* cautarea unui nod dorit */
struct NOD *cauta_nod(struct NOD *tmp, int x){
    if (tmp!=NULL){
        if (x==tmp->x){
            cout<<"Nodul a fost gasit. \n"; return tmp;
        }
        else if (x<tmp->x)
        return cauta_nod(tmp->NOD_stanga, x);
        else return cauta_nod(tmp->NOD_dreapta, x);
    }
    else {
        cout<<"Nodul dorit nu exista in arbore.\n";
```

```
        return NULL;
    }
}
/* programul principal */
int main(){
    struct NOD *prim=NULL, *nod_gasit;
    char operatie; int x;
    /* crearea meniului de optiuni */
    cout<<"MENIU: \n";
    cout<<"\t[1] Inserare nod in arbore \n";
    cout<<"\t[2] Afisare arbore preordine \n";
    cout<<"\t[3] Afisare arbore inordine \n";
    cout<<"\t[4] Afisare arbore postordine \n";
    cout<<"\t[5] Stergere arbore \n";
    cout<<"\t[6] Cautare nod in arbore \n";
    cout<<"\t[0] Iesire din program \n";
    do{
        cout<<"\nIntroduceti operatie: ";
        operatie=getche(); cout<<"\n";
        switch (operatie){
            case '1':
            cout<<"#Inserare nod in arbore# \n";
             cout<<"Introduceti valoarea nodului care va fi in-
serat: ";
            cin>>x; prim=inserare_nod(prim, x); break;
            case '2':
            cout<<"#Afisare arbore preordine# \n";
            if (prim==NULL) cout<<"Atentie: Arbore gol.";
            else afisare_preordine(prim); break;
            case '3':
            cout<<"#Afisare arbore inordine# \n";
            if (prim==NULL) cout<<"Atentie: Arbore gol.";
            else afisare_inordine(prim); break;
            case '4': cout<<"Afisare arbore postordine: \n";
            if (prim==NULL) cout<<"Atentie: Arbore gol.";
            else afisare_postordine(prim); break;
            case '5':
            cout<<"#Stergere arbore# \n";
            if (prim==NULL) cout<<"Atentie: Arbore gol.";
            else{
                 cout<<"Introduceti valoarea nodul al carui ar-
bore va fi sters: ";
```

```
                cin>>x; nod_gasit=cauta_nod(prim, x);
                if (nod_gasit!=NULL){
nod_gasit->NOD_stanga=stergere_arbore(nod_gasit->NOD_stanga);
nod_gasit->NOD_dreapta=stergere_arbore(nod_gasit->NOD_dreapta);
                    cout<<"Arborele a fost sters. \n";
                }
            }
            break;
            case '6':
            cout<<"#Cautare nod in arbore# \n";
            if (prim==NULL) cout<<"Atentie: Arbore gol.";
            else {
                cout<<"Introduceti valoarea nodului: ";
                cin>>x; cauta_nod(prim, x);
            }
            break;
            case '0':
            cout<<"Iesire din program \n";
            stergere_arbore(prim);
            system("PAUSE"); return 0; break;
            default:
            cout<<"Operatie invalida \n";
        }
    }
    while(1);
}
```

Prezentarea execuției secvenței de cod C++ (model):

```
MENIU:
        [1] Inserare nod in arbore
        [2] Afisare arbore preordine
        [3] Afisare arbore inordine
        [4] Afisare arbore postordine
        [5] Stergere arbore
        [6] Cautare nod in arbore
        [0] Iesire din program

Introduceti operatie:
```

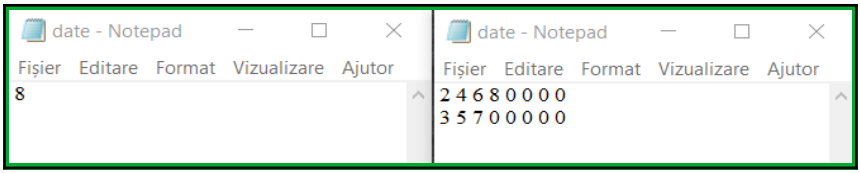
Problema 2 – Parcurgerea pe nivele

Se dă un număr natural n. Construiți un arbore binar complet cu vârfurile 1,2,...,n astfel încât în urma parcurgerii pe nivele să fie afișate valorile 1,2,...,n. Se vor afișa vectorii S și D.

Prezentarea soluției problemei în limbajul C++

```
#include<fstream>
#include<cmath>
using namespace std;
ifstream fin("date.in");
ofstream fout("date.out");
int i,n,S[100],D[100];
/* programul principal */
int main(){
    fin>>n;
    /* construirea arborelui binar conform cerintei */
    int k=1;
    for(i=1;i<=n;i++)
    for(i=2;i<=n;i=i+2){
        S[k]=i;
        if(i<n) D[k]=i+1; k++;
    }
    /* afisarea vectorilor S si D */
    for(i=1;i<=n;i++) fout<<S[i]<<" "; fout<<endl;
    for(i=1;i<=n;i++) fout<<D[i]<<" ";
    fin.close(); fout.close(); return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):



Problema 3 – Forma poloneză cu tablou unidimensional

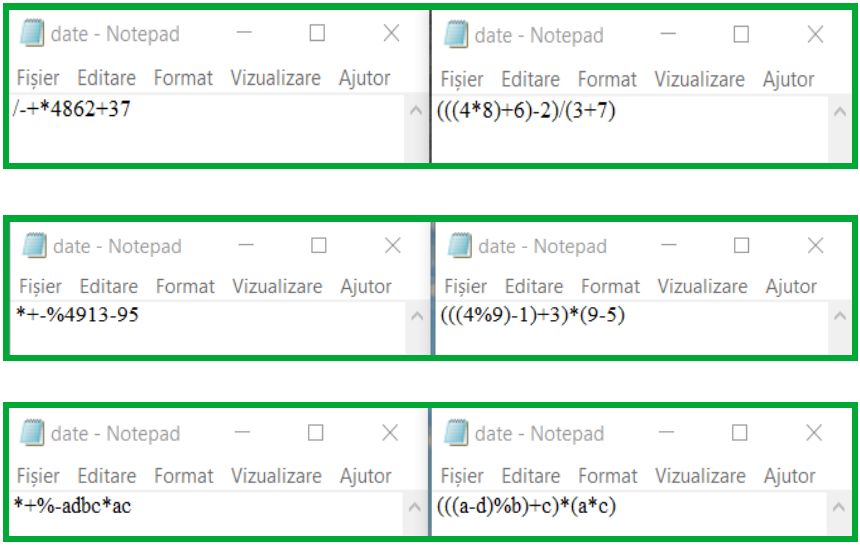
*Se dă o expresie aritmetică în forma poloneză prefixată. Expresia este formată din operatorii + - / * %, iar operanzii sunt dintr-un singur caracter. Construiti arborele binar asociat expresiei citite și afișați expresia în forma normală (infixată). Nu se va utiliza nicio structură.*

Prezentarea soluției problemei în limbajul C++

```
#include <fstream>
#include <cstring>
using namespace std;
ifstream fin("date.in");
ofstream fout("date.out");
int S[100],D[100],T[100],n;
char A[100];
/* parcurgerea elementelor arborelui in inordine */
void SRD(int v){
    if(strchr("+/-*%",A[v]) && v!=0) fout<<"(";
    if(S[v]) SRD(S[v]); fout<<A[v];
    if(D[v]) SRD(D[v]);
    if(strchr("+/-*%",A[v]) && v!=0) fout<<")";
}
/* programul principal */
int main(){
    int n,v,pus;
    fin>>A;
    /* conctruirea arborelui binar asociat expresiei citite */
    n=strlen(A)-1; T[0]=-1;
    for(int i=1;i<strlen(A);i++){
        pus=0; v=0;
        while (!pus){
            while (S[v] && strchr("/*-+%",A[S[v]]))
                  v=S[v];
                  if(S[v]==0){ S[v]=i; pus=1; T[i]=v; }
                  else if(D[v] && strchr("/*-+%",A[D[v]]))
                  v=D[v];
                  else {
                      if(D[v]==0) { D[v]=i; pus=1;T[i]=v;}
                      else {
```

```
        v=T[v];
        while(T[v]!=-1 && !(D[v]==0 || strchr("/*-+%",A[D[v]])))
        v=T[v];
        if(D[v]==0) { D[v]=i; pus=1;T[i]=v;}
        else if(strchr("/*-+%",A[D[v]])) v=D[v];
                }
             }
          }
    }
    SRD(0);
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):



Problema 4 - Forma poloneză cu structură

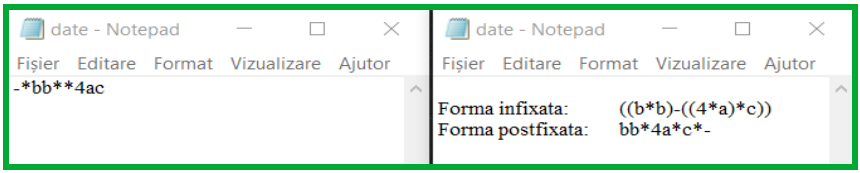
*Se dă o expresie aritmetică în forma poloneză prefixată. Expresia este formată din operatorii + - / * %, iar operanzii sunt litere mici. Afișați expresia în forma normală (infixată) și în forma postfixată. Se va utiliza o structură.*

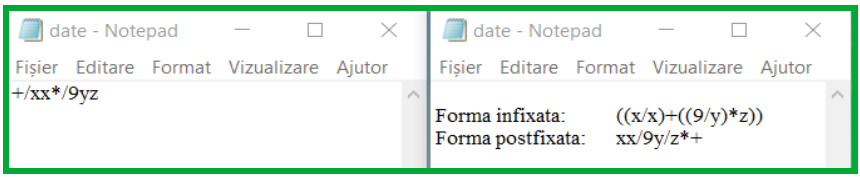
Prezentarea soluției problemei în limbajul C++

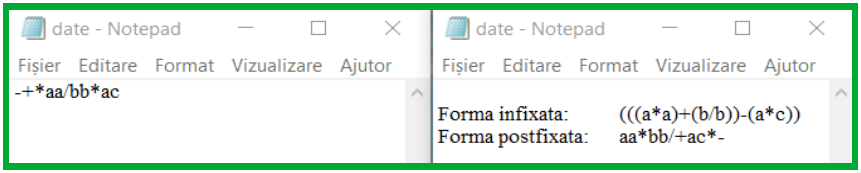
```
#include <fstream>
#include <cstring>
using namespace std;
ifstream fin("date.in");
ofstream fout("date.out");
char op[]="+-*/%";
/* crearea structii nod */
struct nod{
    char info;
    nod *st; nod *dr;
};
/* citirea datelor din fisierul de intrare */
void read(nod* &r){
    char x;
    fin>>x;
    if(strchr(op,x)==NULL){
        r=new nod; r->info=x;
        r->st=NULL; r->dr=NULL;
    }
    else {
        r=new nod; r->info=x;
        read(r->st); read(r->dr);
    }
}
/* parcurgerea elementelor arborelui in inordine */
void SRD(nod *r){
    if(strchr(op,r->info)) fout<<"(";
    if(r->st!=NULL) SRD(r->st);
    fout<<r->info;
    if(r->dr!=NULL) SRD(r->dr);
    if(strchr(op,r->info)) fout<<")";
}
```

```
/* parcurgerea elementelor arborelui in postordine */
void SDR(nod *r){
    if(r->st!=NULL) SDR(r->st);
    if(r->dr!=NULL) SDR(r->dr);
    fout<<r->info;
}
/* programul principal */
int main(){
    nod *r;
    read(r);
    fout<<endl;
    fout<<"Forma infixata: \t"; SRD(r); fout<<endl;
    fout<<"Forma postfixata: \t"; SDR(r); fout<<endl;
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):







Problema 5 – Angajați CEITI

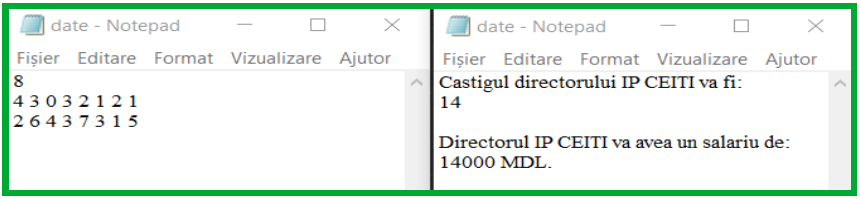
În IP CEITI sunt n angajați, numerotați de la 1 la n, fiecare angajat având un singur șef direct, cu excepția directorului, care nu are sef. Ierarhia instituției este dată printr-un vector de tip tată. Fiecare angajat al instituției are un salariu dat printr-un număr natural. Angajații și seful sunt recompensați astfel: câștigul fiecărui salariat este egal cu salariul său la care se adaugă media aritmetică a câștigurilor subordonaților săi direcți. Media aritmetică se rotunjeste prin adaos la un număr întreg (de exemplu 5.33 se rotunjeste la 6). Angajații care nu au subordonați direcți câștigă doar salariul. Calculați care este câștigul directorului instituției, dacă salariul se estimează a fi calculat în mii de lei.

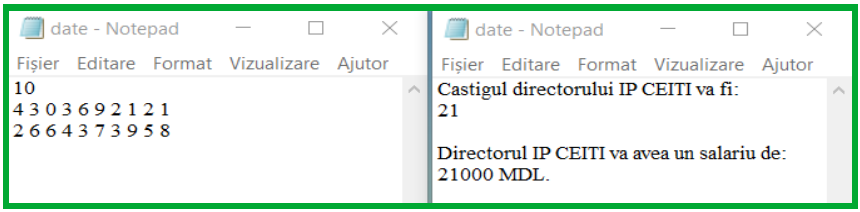
Prezentarea soluției problemei în limbajul C++

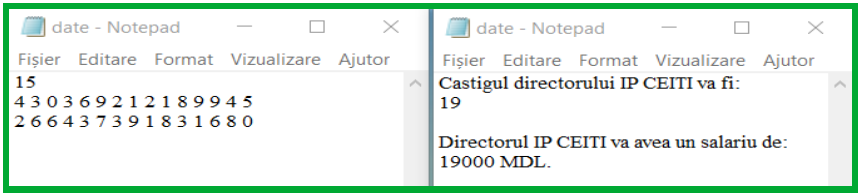
```
#include <fstream>
#include <cmath>
using namespace std;
ifstream fin("date.in");
ofstream fout("date.out");
int r,n, t[101],x[101], s[101];
int fii(int r){
    int k=0;
    for(int i=1;i<=n;i++) if(t[i]==r) k++;
    return k;
}
/* functia de calcul a salariului */
int calcul(int v){
    s[v]=0;
    for(int i=1;i<=n;i++)
    if(t[i]==v) s[v]=s[v]+calcul(i);
    if(fii(v)>0) return (x[v]+ceil((float)s[v]/fii(v)));
    else return x[v];
}
/* programul principal */
int main(){
    /* citirea datelor din fisierul de intrare */
    fin>>n;
    for(int i=1;i<=n;i++){
        fin>>t[i];
```

```
        if(t[i]==0) r=i;
    }
    for(int i=1;i<=n;i++) fin>>x[i];
    /* afisarea castigului directorului */
    fout<<"Castigul directorului IP CEITI va fi: \n";
    fout<<calcul(r);
    fout<<"\n\n";
    fout<<"Directorul IP CEITI va avea un salariu de: \n";
    fout<<1000*calcul(r)<<" MDL. \n";
    fin.close();
    fout.close();
    return 0;
}
```

Prezentarea execuției secvenței de cod C++ (model):







7.5 Modele de probleme propuse

1. Elaborați subprograme în C++ pentru următoarele probleme simple.

a. *Să se realizeze un program ce permite implementarea unui arbore binar de căutare. Programul permite afișarea pe ecran a unui meniu cu următoarele operații posibile:*
- *Să se calculeze numărul de frunze din arbore.*
- *Să se calculeze numărul maxim de niveluri din arbore.*
- *Să se șteargă complet arborele pornind de la rădăcina acestuia.*

b. *Să se realizeze un program ce permite implementarea unui arbore binar de căutare. Implementați operațiile de afișare, ștergere și căutare a nodurilor unui arbore folosind funcții nerecursive.*

2. Elaborați un program în C++ pentru următoarea problemă cu fișiere.

a. *Se dă un număr natural n. Construiți un arbore binar complet cu vârfurile 1,2...n astfel încât în urma parcurgerii pe nivele să fie afișate valorile 1,2...n. Se vor afișa vectorii S și D.*
Exemplu: ***n=8***
Soluție: ***S=2 4 6 8 0 0 0 0 0*** *și* ***D=3 5 7 9 0 0 0 0 0***

b. *Se dă o expresie aritmetică în forma normală (infixată). Expresia este formată din operatorii „+”, „-”, „/”, „*” și „%”, iar operanzii sunt litere mici. Afișați expresia în forma poloneză prefixată și în forma postfixată.*
Exemplu: ***((x*x)-((4*y)*z))***
Soluție: forma prefixată: ***-*xx**4yz*** *și forma postfixată:* ***xx*4y*z*-***

c. *Se dă un arbore cu n vârfuri prin vectorul TATA. Afișați care dintre vârfurile arborelui vor fi alese ca rădăcină a acestuia astfel încât arborele să aibă înălțimea minimă.*
Exemplu: ***n=6*** *și* ***vectorul TATA este 2 0 2 1 3 1***
Soluție: ***2***

3. Elaborați un program în C++ pentru următoarele probleme.

3.1 *Se citește un arbore prin vectorul de tați. Să se determine și să se afișeze cel mai lung lanț din arbore.*
Exemplu: Pentru vectorul de tați ***2 0 2 5 2 5 3 7***
Soluție: Se afișează lanțul: ***4 5 2 3 7 8***

3.2 *Se dau în fișierul date.in numerele n, p și q, unde n este numărul de noduri ale unui arbore, iar p și q sunt noduri din arbore. Apoi se citește vectorul de tați prin care este reprezentat arborele. Afișați lanțul elementar de la vârful p la vârful q.*

Date de intrare	Date de ieșire
7 2 5 4 1 7 0 7 7 1	2 1 7 5

3.3 *Se dau doi arbori cu rădăcină prin doi vectori de tip tată. Determinați dacă cei doi arbori cu rădăcina reprezintă același arbore (diferă doar în urma alegerii rădăcinilor diferite).*

Date de intrare	Date de ieșire
5 0 1 1 2 2 3 1 0 2 2	DA

3.4 *Se citește un vector A cu n elemente numere întregi. Plasați indicii 1,2...n într-un arbore binar astfel încât în urma parcurgerii în inordine a aceastuia să se afișeze vectorul A sortat crescător. Arborele binar va fi reprezentat prin vectorii S și D.*

Date de intrare	Date de ieșire
n=7 A=3 4 1 2 9 0 6	A=0 1 2 3 4 6 9 S=3 0 6 0 7 0 0 D=2 5 4 0 0 0 0

7.6 Portofoliul elevului pentru arbori

1. *Se dă o expresie aritmetică în forma poloneză infixată (normală). Expresia este formată din operatorii „+”, „-”, „/”, „*”, „%”, iar operanzii sunt litere mici. Afișați expresia în forma poloneză prefixată și postfixată. Fiecare elev din subgrupă va realiza câte un subprogram pentru inplementarea celor două forme poloneze, fiecare elev va avea o expresie individuală conform numărului de ordine.*

Nr.	Secvența de numere	Nr.	Secvența de numere
1	*(a-b)*(a+b)-(a*a-b*b)*	**11**	*(a+b)*(a*a-a*b+b*b)-(a*a*a+b*b*b)*
2	*a*(b+c)-(a*b+a*c)*	**12**	*(a-b)*(a*a+a*b+b*b)-(a*a*a-b*b*b)*
3	*a*(b-c)-(a*b-a*c)*	**13**	*(a+b)*(a*a-a*b+b*b)+(a*a*a+b*b*b)*
4	*(a-b)*(a-b)-(a*a-2*a*b+b*b)*	**14**	*(a-b)*(a*a+a*b+b*b)+(a*a*a-b*b*b)*
5	*(a+b)*(a+b)-(a*a+2*a*b+b*b)*	**15**	*(a+b)*(c+d)%(a-b)/(c+d)*
6	*(a-b)*(a+b)+(a*a-b*b)*	**16**	*(a-b)*(a-b)%(a*a-2*a*b+b*b)*
7	*a*(b+c)+(a*b+a*c)*	**17**	*(a+b)*(a+b)%(a*a+2*a*b+b*b)*
8	*a*(b-c)+(a*b-a*c)*	**18**	*(a-b)*(a+b)/(a*a-b*b)*
9	*(a-b)*(a-b)+(a*a-2*a*b+b*b)*	**19**	*a*(b+c)/(a*b+a*c)*
10	*(a+b)*(a+b)+(a*a+2*a*b+b*b)*	**20**	*a*(b-c)/(a*b-a*c)*

2. *Din fișierul arbore.in se citește un număr natural n reprezentand numărul de vârfuri ale unui arbore binar și apoi vectorii S și D pentru descendenții fiecărui nod din arbore.*

- ***a.*** *Să se parcurgă arborele în preordine, inordine și postordine;*
- ***b.*** *Să se parcurgă arborele pe nivele;*
- ***c.*** *Să se calculeze și să se afișeze înălțimea arborelui.*

Fiecare elev din subgrupă va realiza câte un subprogram pentru inplementarea celor trei operații, fiecare elev va avea de citit dintr-un fișier datele despre un arbore individual conform numărului de ordine.

Nr.	Informația din fișier	Nr.	Informația din fișier	Nr.	Informația din fișier
1	7 2 4 0 0 7 0 0 3 5 6 0 0 0 0	**11**	7 21 28 27 0 0 0 0 19 16 0 34 0 0 0	**21**	7 8 7 5 4 0 0 0 9 3 0 2 0 0 0
2	7 9 6 0 2 0 0 0 7 1 4 8 0 0 0	**12**	7 43 46 30 17 0 0 0 18 24 42 22 0 0 0	**22**	7 9 6 8 2 0 0 0 0 4 5 7 0 0 0
3	7 3 5 6 0 0 0 0 2 4 0 0 7 0 0	**13**	**7** 43 10 47 24 0 0 0 30 14 50 34 0 0 0	**23**	7 6 3 1 4 0 0 0 7 0 5 8 0 0 0
4	7 2 8 5 9 0 0 0 1 4 0 7 0 0 0	**14**	**7** 45 35 16 0 0 0 0 26 37 0 20 0 0 0	**24**	7 3 1 2 0 0 0 0 6 4 9 5 0 0 0
5	7 3 8 6 0 0 0 0 2 4 0 1 7 0 0	**15**	**7** 21 49 15 12 0 0 0 19 46 32 36 0 0 0	**25**	7 8 7 6 4 0 0 0 0 5 3 1 0 0 0
6	7 2 4 0 1 7 0 0 3 8 6 0 0 0 0	**16**	**7** 27 48 28 18 0 0 0 44 29 33 42 0 0 0	**26**	7 5 9 8 6 0 0 0 1 3 4 0 0 0 0
7	7 2 4 0 7 0 0 0 5 8 6 3 0 0 0	**17**	**7** 31 17 40 39 0 0 0 22 11 13 41 0 0 0	**27**	7 7 0 9 5 0 0 0 3 6 2 1 0 0 0
8	7 6 3 0 4 0 0 0 2 8 5 0 0 0 0	**18**	7 10 47 0 25 0 0 0 19 30 42 0 0 0 0	**28**	7 1 9 6 3 0 0 0 4 0 7 8 0 0 0
9	7 4 6 9 5 0 0 0 1 8 0 2 0 0 0	**19**	7 12 39 33 32 0 0 0 43 29 22 49 0 0 0	**29**	7 4 5 7 1 0 0 0 9 2 8 6 0 0 0
10	7 7 8 4 3 0 0 0 1 6 5 2 0 0 0	**20**	7 38 28 35 18 0 0 0 23 50 14 45 0 0 0	**30**	7 1 5 6 8 0 0 0 7 2 9 0 0 0 0

7.7 Model de test grilă pentru arbori

Nr.	Conținutul	Punctaj
1	*Arborii sunt structuri de date logice şi omogene.* ◯ adevărat ◯ fals	*2p*
2	*Arborii binari sunt arbori în care nodurile conţin cel mult doi descendenţi.* ◯ adevărat ◯ fals	*2p*
3	*Fiecare nod din arbore conţine informaţiile utile, un întreg care reţine numărul de fii şi un vector de pointeri către fii.* ◯ adevărat ◯ fals	*2p*
4	*Operaţia de ștergere a unui nod presupune alocarea memoriei pentru acesta, copierea informaţiei utile şi modificarea legăturii părintelui.* ◯ adevărat ◯ fals	*2p*
5	*Arborii binari de căutare sunt arbori binari în care nodurile sunt memorate ordonat în funcţie de o cheie.* ◯ adevărat ◯ fals	*2p*
6	*Arborii binari de căutare permit regăsirea rapidă a informaţiilor într-un timp:* ◯ O (log2 N) ◯ O (log 2*N) ◯ O (log3 N) ◯ O (log 3*N)	*2p*
7	*Fie că avem parcurgerea în lățime a unui arbore binar reprezentat prin nodurile: 1, 2, 3, 4, 5, 6, 7, 8. Stabiliți care este parcurgerea corectă a arborelui în preordine.* ◯ 4 2 5 1 6 3 7 8 ◯ 4 5 2 6 8 7 3 1 ◯ 1 2 4 5 3 6 7 8 ◯ alt răspuns	*2p*
8	*Ce presupune parcurgerea în postordine?* ◯ S-V-D ◯ S-D-V ◯ V-S-D ◯ alt răspuns	*2p*

9	*Câte modalităţi de parcurgere a elementelor unui arbore binar există ?* ◯ 3 ◯ 4 ◯ 2 ◯ alt răspuns	*2p*
10	*Care este tipul de parcurgere al unui arbore binar asociat unei expresii aritmetice pentru a obţine forma poloneză ?* ◯ inordine ◯ postordine ◯ preordine ◯ lățime	*2p*
11	*Care este forma poloneză asociată expresiei: a*(b+c)-e/(a+d)+h ?* ◯ abc+*ead+/-h+ ◯ a*b+c-e/a+d+ ◯ -+h*/a+e+bcad ◯ +-*a+bc/c+ad	*2p*
12	*Care este forma poloneză pentru expresia (a+b)*(a+c)*(b+c) ?* ◯ a+b*a+c*b+c ◯ **+++bcabac ◯ **+ab+ac+bc ◯ ab+ac+*bc+*	*2p*
13	*Un arbore cu 11 noduri, numerotate de la 1 la 11, este memorat cu ajutorul vectorului de „taţi” t=(2, 5, 5, 3, 0, 2, 4, 6, 6, 2, 3). Mulţimea tuturor ascendenţilor nodului 8 este:* ◯ 1 2 5 6 10 ◯ 6 ◯ 6 2 5 ◯ 5 2	*2p*
14	*Care dintre următorii vectori NU poate reprezenta vectorul „de taţi” al unui arbore cu rădăcină, cu 5 noduri, numerotate de la 1 la 5?* ◯ 3 1 0 1 2 ◯ 3 4 0 2 3 ◯ 2 0 1 1 1 2 ◯ 4 1 1 0 2	*2p*

Barem de notare propus

Nota	10	9	8	7	6	5	4	3	2
%	100-95	94-88	87-78	77-63	62-48	47-33	32-21	20-10	9-5
Punctaj	28-27	26-25	24-22	21-18	19-13	12-9	8-6	5-3	2-1

BIBLIOGRAFIE

Resurse bibliografice clasificate descendent conform anului publicației.

Nr.	Autorul şi titlul resursei fundamentale
1	*Changkun O., " Modern C++ tutorial: C++11/ 14/ 17/ 20/ on the fly ", 2020, pag. 89;*
2	*Аленский Н. А. , Мармыш Д. Е. , "Сборник задач. Методы программирования. ", 2018, с. 98;*
3	*Galowicz J., "C++17 STL Cookbook", 2017, pag. 589;*
4	*Runceanu A., Runceanu M., "Structuri de date alocate dinamic", Ed. Academica Brâncusi din Târgu Jiu, 2016, pag. 212;*
5	*Meyers S., "Effective modern C++", 2015, pag. 334;*
6	*Gremalschi A., "Informatică" – Manual pentru clasa a XI-a, Ed. Știința, 2014;*
7	*Shaffer C. A., "Data structures and algorithm analysis (Ed. 3.2, C++ Version)", 2013, pag. 615;*
8	*Neciu I., "Culegere de probleme rezolvate în C/C++", Ed. Sfântul Ierarh Nicolae, 2011, pag. 76;*
9	*Goodrich M. T., Tamassia R., Mount D. M., "Data structures and algorithms in C++" (second edition), 2011, pag. 738;*
10	*Morin P., "Open data structures in C++", 2011, pag. 264;*
11	*Malik D. S., "Data structures using C++", 2010, pag. 945;*
12	*Пышкин Е. В., "Структуры данных и алгоритмы: реализация на C/C++", 2009, с. 200;*
13	*Поляков К., "Программирование на языке Си/Си++", 2009, с. 230;*
14	*Аксёнова Е. А. , Соколов А. В. , "Алгоритмы и структуры данных на C++" , Издательство ПетрГУ, 2008, с. 82;*
15	*Das V. V., "Principles of data structures using C and C++", 2006, pag. 375;*

Nr.	Autorul şi titlul resursei fundamentale
16	*Chiriță P., Stratan C., Opincaru C., "Culegere de probleme în C/C++", Ed. Niculescu SRL, 2003, pag. 256;*
17	*Weiss M. A., "Data structures and problem solving using C++", 2003, pag. 977;*
18	*Meyers S., "Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library", 2001, pag. 288;*
19	*Kruse R. L., Ryba A. J., "Data structures and program design in C++", 2000, pag. 734;*
20	*Josuttis N. M., "The C++ Standard Library", 2000, pag. 642;*

Resurse web utilizate și recomandate.

Nr.	Titlul resursei recomandate
1	*C++ Programming Language* **https://www.geeksforgeeks.org**
2	*Cristian Ioniță – Structuri de date* **http://ase.softmentor.ro/**
3	*Open Course Ware - Structuri de date* **https://ocw.cs.pub.ro/courses/sd-ca**
4	*C++ STL: Tutorial and Examples* **http://www.yolinux.com/TUTORIALS/LinuxTutorialC++STL.html**
5	*Website devoted to teaching you how to program in C++* **https://www.learncpp.com**
6	*Standard Template Library in C++* **https://www.studytonight.com/cpp/stl/**
7	*STL: стандартная библиотека шаблонов C++* **https://tproger.ru/articles/stl-cpp/**
8	*Angular 2: что это такое?* **https://codelessons.ru/vse-stati**

ANEXE

1. Elaborarea unui fișier antet (header) în C++

*C++ oferă utilizatorilor săi o varietate de funcții, care sunt incluse în fișierele antet. În C++, toate fișierele antet se pot termina sau nu cu extensia „**.h**”, dar în C, toate fișierele antet trebuie să se încheie în mod necesar cu extensia „**.h**”.*

Un fișier antet conține:

- *Definiții ale funcției (subprogram);*
- *Definiții ale tipului de date;*
- *Macrocomenzi.*

*Aceste fișiere antet pot fi importate în programul C++ cu ajutorul unei directive preprocesator „**#include**”. Această directivă de preprocesare este utilizată pentru instruirea compilatorului.*

Un program C, trebuie să conțină în mod necesar fișierul antet care reprezintă intrarea și ieșirea standard utilizate pentru a lua intrarea cu ajutorul funcțiilor scanf () și respectiv printf ().

În programul C++ există fișierul antet care reprezintă fluxul de intrare și ieșire folosit pentru a lua intrarea cu ajutorul „cin” și respectiv „cout”.

Există două tipuri de fișiere antet:

- *Fișiere antet preexistente (fișierele care sunt deja disponibile în compilatorul C/C++, deci trebuie doar să le importăm);*
- *Fișiere antet definite de utilizator (aceste fișiere sunt definite de utilizator și pot fi importate folosind directiva „#include”).*

Sintaxă: #include <filename.h> *sau* #include "filename.h"

Putem include fișiere antet în programul nostru folosind una din sintaxele de mai sus, indiferent dacă este un fișier antet predefinit sau definit de utilizator.

Preprocesatorul „#include” este responsabil pentru direcționarea compilatorului că fișierul antet trebuie procesat înainte de compilare și include toate tipurile de date și definițiile funcției necesare.

De ce avem nevoie de subprograme (funcţii) ?

- *Funcțiile ne ajută la reducerea redundanței codului. Dacă funcționalitatea este realizată în mai multe locuri din software, atunci creăm o funcție și o utilizăm la necesitate. Aceasta contribuie la întreținerea și optimizarea codului sursă.*
- *Funcțiile contribuie la modularea codului Luați în considerare un fișier mare care are multe linii de coduri. Devine foarte simplu să citești și să folosești codul dacă codul este împărțit în funcții.*
- *Funcțiile oferă abstractizare. De exemplu, putem folosi funcțiile bibliotecii fără a ne îngrijora de funcționarea lor internă.*

Toate variabilele utilizează tipul de date în timpul declarației pentru a restricționa tipul de date care trebuie stocat. Prin urmare, putem spune că tipurile de date sunt utilizate pentru a indica variabilelor tipul de date pe care le poate stoca.

Ori de câte ori o variabilă este definită în C++, compilatorul alocă o anumită memorie pentru acea variabilă pe baza tipului de date cu care este declarată. Fiecare tip de date necesită o cantitate diferită de memorie.

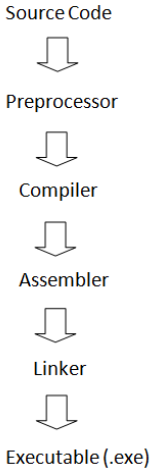
Tipurile de date în C ++ sunt împărţite în principal în trei tipuri:

1. Tipuri de date primitive: Aceste tipuri de date sunt tipuri de date încorporate sau predefinite și pot fi utilizate direct de către utilizator pentru a declara variabile. exemplu: int, char, float, bool etc. Tipurile de date primitive disponibile în C++ sunt: Integer, Character, Boolean, Floating Point, Double Floating Point, Valueless or Void, Wide Character.

2. Tipuri de date derivate: tipurile de date derivate din tipurile de date primitive sau încorporate sunt denumite tipuri de date derivate. Acestea pot fi de patru tipuri și anume: Function, Array, Pointer, Reference.

3. Tipuri de date abstracte sau definite de utilizator: aceste tipuri de date sunt definite de utilizator însuși. De exemplu, definirea unei clase în C++ sau a unei structuri. C++ oferă următoarele tipuri de date definite de utilizator:Class, Structure, Union, Enumeration, Typedef defined DataType.

Preprocesoarele



După cum sugerează și numele, preprocesatoarele sunt programe care procesează codul-sursă înainte de compilare. Există o serie de pași implicați între scrierea unui program și executarea unui program în C sau C++. Să aruncăm o privire la acești pași înainte de a începe să aflăm efectiv despre preprocesori.

Codul sursă scris de programatori este stocat în fișierul ***program.c****. Acest fișier este apoi procesat de preprocesoare și se generează un fișier de cod sursă extins numit program. Acest fișier extins este compilat de compilator și se gene-rează un fișier de cod obiect numit* ***program.obj****. În cele din urmă, linkerul leagă acest fișier de cod obiect cu codul obiect al funcțiilor bibliotecii pentru a genera fișierul executabil* ***program.exe****.*

Programele de preprocesare oferă directive de preprocesare care îi spun compilatorului să preproceseze codul sursă înainte de compilare.

Toate aceste directive de preprocesare încep cu simbolul „#" (hash). Simbolul „#" indică faptul că, orice instrucțiune începe cu #, se îndreaptă către programul preprocesator, iar programul preprocesor va executa această instrucțiune. Exemple de directive ale preprocesorului sunt: #include, #define, #ifndef, etc. Spre exemplu, directiva #include va include cod suplimentar pentru programul dvs. Putem plasa aceste directive preprocesor oriunde în programul nostru.

Macro-urile

Când rulează compilatorul C++, apelează mai întâi preprocesorul pentru a găsi directivele compilatorului care pot fi incluse în codul sursă.

Procesul de compilare folosește preprocesorul pentru a compila codul sursă, un asamblator pentru a traduce acest lucru în codul mașinii și un linker pentru a converti unul sau mai multe obiecte binare într-un program executabil.

Macro comenzile sunt o bucată de cod dintr-un program căruia i se dă un anumit nume. Ori de câte ori acest nume este întâlnit de compilator, compilatorul înlocuiește numele cu codul real. Directiva „#define" este utilizată pentru a defini un macro.

2. Analiza codului dintre listele liniare și listele circulare

Partea 1 – Listele liniare simplu înlănţuite şi cele dublu înlănţuite (A)

Structura dinamică - LLSÎ	Structura dinamică - LLDÎ
Definirea unui nod (1)	

`typedef struct nod{`
`  int inf;`
`  nod *urm;`
`} *pnod;`
	`typedef struct nod{`
	`  int inf;`
	`  nod *urm,*ant;`
	`} *pnod;`

Adăugare nod la începutul listei (2)	

`void adinceput(int x){`
`  pnod p; p=new nod;`
`  p->inf=x; p->urm=cap;`
`  if(cap==NULL) sf=p;`
`  cap=p;`
`}`
	`void adinceput(int x){`
	`  pnod p; p=new nod;`
	`  p->inf=x; p->urm=cap;`
	`  p->ant=NULL;`
	`  if(cap==NULL) sf=p;`
	`  else cap->ant=p;`
	`  cap=p;`
	`}`

Adăugare nod la sfârşitul listei (3)	

`void adsfarsit(int x){`
`  pnod p; p=new nod;`
`  p->inf=x; p->urm=NULL;`
`  if(cap==NULL) cap=p;`
`  else sf->urm=p;`
`  sf=p;`
`}`
	`void adsfarsit(int x){`
	`  pnod p; p=new nod;`
	`  p->inf=x; p->urm=NULL;`
	`  p->ant=sf;`
	`  if(cap==NULL) cap=p;`
	`  else sf->urm=p;`
	`  sf=p;`
	`}`

Adăugare nod după un careva nod al listei (4)	

`void ad_dupa(int x, int y){`
`  pnod p,q; p=cap;`
`  while((p)&&(p->inf!=x))`
`  p=p->urm;`
`  if(p) {`
`  q= new nod; q->inf=y;`
`  q->urm=p->urm; p->urm=q;`
`  if(q->urm==NULL) sf=q;`
	`void ad_dupa(int x, int y){`
	`pnod p,q; p=cap;`
	`while((p!=NULL)&&(p->inf!=x))`
	`p=p->urm;`
	`if(p!=NULL) {`
	`  q= new nod; q->inf=y;`
	`  q->urm=p->urm; q->ant=p;`
	`  p->urm=q;`

Structura dinamică - LLSÎ	Structura dinamică - LLDÎ

}
}
	if(q->urm!=NULL)
	q->urm->ant=q; else sf=q;
	}
	}

Adăugare nod înaintea unui careva nod al listei (5)	

void ad_inainte(int x, int y{
pnod p,q; p=cap;
while((p!=NULL)&&(p->inf!=x))
p=p->urm;
if(p!=NULL){
q= new nod; q->inf=p->inf;
p->inf=y; q->urm=p->urm;
p->urm=q;
if (q->urm==NULL) sf=q;
}
}
	void ad_inainte(int x, int y{
	pnod p,q; p=cap;
	while((p!=NULL)&&(p->inf!=x))
	p=p->urm;
	if(p!=NULL){
	q= new nod; q->inf=y;
	q->urm=p; q->ant=p->ant;
	if (q->ant!=NULL)
	q->ant->urm=q; else cap=q;
	p->ant=q;
	}
	}

Ştergerea primului nod al listei (6)	

void stergecap(void){
pnod p; p=cap;
cap=cap->urm;
if (cap==NULL) sf=NULL;
delete p;
}
	void stergecap(void){
	pnod p; p=cap;
	cap=cap->urm;
	if (cap==NULL) sf=NULL;
	else cap->ant=NULL;
	delete p;
	}

Ştergerea nodui curent al listei (7)	

void stergenod(int x){
pnod p,q; p=cap; q=NULL;
while(p&&(p->inf!=x)){
q=p; p=p->urm;
}
if(p!=NULL) {
if(p==s){
q->urm=NULL;
if(p==cap) cap=NULL;
sf=q; delete(p);
} else{
	void stergenod(int x){
	pnod p; p=cap;
	while(p&&(p->inf!=x))
	p=p->urm;
	if(p) {
	if(p->ant)
	p->ant->urm=p->urm;
	else cap=p->urm;
	if(p->urm)
	p->urm->ant=p->ant;
	else sf=p->ant;

Structura dinamică - LLSÎ	Structura dinamică - LLDÎ

q=p->urm;
p->inf=q->inf;
p->urm=q->urm;
if (p->urm==NULL) sf=p;
delete q;
}
}
	delete p;
	}
	}

Ştergerea intregii liste (8)	

void stergelista(void){
pnod p,q; p=cap;
cap=NULL;
while (p!=NULL){
q=p; p=p->urm; delete q;
}
}
	void stergelista(void){
	pnod p,q; p=cap;
	cap=NULL; sf=NULL;
	while (p!=NULL){
	q=p; p=p->urm; delete q;
	}
	}

Căutarea unui nod într-o listă (9)	

pnod cauta(int x){
pnod p; p=cap;
// trebuie ca p!=NULL
while ((p) && (p->inf!=x))
p=p->urm;
return p;
}
	pnod cauta(int x){
	pnod p; p=cap;
	// trebuie ca p!=NULL
	while ((p) && (p->inf!=x))
	p=p->urm;
	return p;
	}

Parcurgerea listei de la început la sfârşit (10)	

void parccap(void){
pnod p; p=cap;
while(p) {
/* o operatie asupra
informatiei nodului */
p=p->urm;
}
}
	void parccap(void){
	pnod p; p=cap;
	while(p) {
	/* o operatie asupra
	informatiei nodului */
	p=p->urm;
	}
	}

Structura dinamică - LCSÎ	Structura dinamică - LCDÎ
Definirea unui nod (1)	

`typedef struct nod{`
`  int inf;`
`  nod *urm;`
`} *pnod;`
	`typedef struct nod{`
	`  int inf;`
	`  nod *urm,*ant;`
	`} *pnod;`

Adăugare nod la începutul listei (2)	

`void adinceput(int x){`
`  pnod p; p=new nod;`
`  p->inf=x;`
`  if(cap==NULL) p->urm=p;`
`  else{`
`  p->urm=cap; q=cap;`
`  while(q->urm!=cap)`
`  q=q->urm; q->urm=p;`
`  }`
`  cap=p;`
`}`
	`void adinceput(int x){`
	`  pnod p,q; p=new nod;`
	`  p->inf=x;`
	`  if(cap==NULL){`
	`  p->urm=p; p->ant=p;`
	`  }`
	`  else{`
	`  q=cap->ant; p->urm=cap;`
	`  cap->ant=p; q->urm=p;`
	`  p->ant=q;`
	`  }`
	`  cap=p;`
	`}`

Adăugare nod înaintea unui careva nod al listei (3)	

`void ad_inainte(int x,int y){`
`  pnod p,q; p=cap;`
`  do {`
`  p=p->urm;`
`  }`
`  // trebuie ca p!=cap`
`  while ((p) && (p→inf!=x));`
`  if(p->inf==x) {`
`  q= new nod;`
`  q->inf=p->inf; p->inf=y;`
`  q->urm=p->urm; p->urm=q;`
`  }`
`}`
	`void ad_inainte(int x,int y){`
	`  pnod p,q; p=cap;`
	`  do {`
	`  p=p->urm;`
	`  }`
	`  // trebuie ca p!=cap`
	`  while ((p) && (p→inf!=x));`
	`  if(p->inf==x) {`
	`  q= new nod;`
	`  q->inf=p->inf; p->inf=y;`
	`  q->urm=p->urm; p->urm=q;`
	`  q->ant=p; q->urm->ant=q;`
	`  }`
	`}`

Structura dinamică - LCSÎ	Structura dinamică - LCDÎ
Adăugare nod după un careva nod al listei (4)	

`void ad_dupa(int x, int y){`
`  pnod p,q; p=cap;`
`  do {`
`  p=p->urm;`
`  }`
`  // trebuie ca p!=cap`
`  while((p)&&(p->inf!=x));`
`  if(p->inf==x) {`
`  q= new nod; q->inf=y;`
`  q->urm=p->urm; p->urm=q;`
`  }`
`}`
	`void ad_dupa(int x, int y){`
	`pnod p,q; p=cap;`
	`do {`
	`  p=p->urm;`
	`  }`
	`  // trebuie ca p!=cap`
	`  while((p)&&(p->inf!=x));`
	`  if(p->inf==x) {`
	`  q= new nod; q->inf=y;`
	`  q->urm=p->urm; q->ant=p;`
	`  p->urm=q; q->urm->ant=q;`
	`  }`
	`}`

Ştergerea primului nod al listei (5)	

`void stergecap(void){`
`  pnod p,q; p=cap;`
`  if (cap=cap->urm) cap=NULL;`
`  else{`
`  q=cap;`
`  while(q->urm!=cap)`
`  q=q->urm; cap=cap->urm;`
`  q->urm=cap;`
`  }`
`  delete p;`
`}`
	`void stergecap(void){`
	`  pnod p,q; p=cap;`
	`  if (cap=cap->urm) cap=NULL;`
	`  else{`
	`  cap=cap->urm;`
	`  cap->ant=p->ant;`
	`  cap->ant->urm=cap;`
	`  }`
	`  delete p;`
	`}`

Ştergerea nodui curent al listei (6)	

`void stergenod(int x){`
`  pnod p,q; p=cap;`
`  do {`
`  p=p->urm;`
`  } // trebuie ca p!=cap`
`  while((p)&&(p->inf!=x));`
`  if(p->inf==x) {`
`  if(p==p->urm) cap=NULL;`
`  else{`
`  q=p;`
`  while(q->urm!=p)`
	`void stergenod(int x){`
	`  pnod p,q; p=cap;`
	`  do {`
	`  p=p->urm;`
	`  } // trebuie ca p!=cap`
	`  while((p)&&(p->inf!=x));`
	`  if(p->inf==x) {`
	`  if(p==p->urm) cap=NULL;`
	`  else{`
	`  q=p->ant;`
	`  if(p==cap)`

Structura dinamică - LCSÎ	Structura dinamică - LCDÎ

q=q->urm;
if(p==cap)
cap=cap->urm;
q->urm=p->urm;
}
delete(p);
}
}
	cap=cap->urm;
	q->urm=p->urm;
	q->urm->ant=q;
	}
	delete(p);
	}

Ştergerea intregii liste (7)	

void stergelista(void){
pnod p,q; p=cap->urm;
cap->urm=NULL;
while (p!=NULL){
q=p; p=p->urm; delete q;
}
}
	void stergelista(void){
	pnod p,q; p=cap->urm;
	cap->urm=NULL; cap=NULL;
	while (p!=NULL){
	q=p; p=p->urm; delete q;
	}
	}

Căutarea unui nod într-o listă (8)	

pnod cauta(int x){
pnod p; p=cap;
do {
p=p->urm;
} // trebuie ca p!=cap
while((p)&&(p→inf!=x));
if(p->inf!=x) return NULL;
else return p;
}
	pnod cauta(int x){
	pnod p; p=cap;
	do {
	p=p->urm;
	} // trebuie ca p!=cap
	while((p)&&(p→inf!=x));
	if(p->inf!=x) return NULL;
	else return p;
	}

Parcurgerea listei de la început la sfârşit (9)	

void parccap(void){
pnod p; p=cap;
do {
/* o operatie asupra informatiei nodului */
}
while(p!=cap);
}
	void parccap(void){
	pnod p; p=cap;
	do {
	/* o operatie asupra informatiei nodului */
	p=p->urm;
	}
	while(p!=cap);
	}

3. Analiza codului dintre stivă și coadă

Structura dinamică - Stiva	Structura dinamică - Coada
Definirea unui nod (1)	

`typedef struct nod{`
`  int inf; nod *urm;`
`} *pnod;`
	`typedef struct nod{`
	`  int inf; nod *urm;`
	`} *pnod;`

Adăugarea unui nod (2)	

`void push(int x){`
`  pnod p; p=new nod;`
`  p->inf=x; p->urm=cap;`
`  cap=p;`
`}`
	`void add(int x){`
	`  pnod p; p=new nod;`
	`  p->inf=x; p->urm=NULL;`
	`  if (cap==NULL) cap=p;`
	`  else sf->urm=p; sf=p;`
	`}`

Extragerea unui nod (3)	

`int pop(void){`
`  pnod p; p=cap;`
`  return p->inf;`
`  cap=cap->urm; delete p;`
`}`
	`int extr(void){`
	`  pnod p; p=cap; return p->inf;`
	`  cap=cap->urm;`
	`  if (cap==NULL) sf=NULL;`
	`  delete p;`
	`}`

O analiză comparativă a acestor două structuri dinamice prestabilite

Structura dinamică - Stiva	Structura dinamică - Coada
■ *Stiva este un tip de date abstract reprezentat de o stivă fizică de entități în care inserarea și ștergerea au loc la același capăt.* ■ *Se bazează pe principiul de funcționare al LIFO, ceea ce înseamnă ultimul intrat primul ieșit.* ■ *Obiectul poate fi adăugat în stivă unul câte unul, ceea ce înseamnă,*	◆ *Coada este, de asemenea, un tip de date abstract similar cu stiva, cu excepția faptului că este deschisă la ambele capete.* ◆ *Se bazează pe principiul de funcționare al FIFO care înseamnă primul intrat primul ieșit.* ◆ *Obiectul care este adăugat mai întâi (primul), poate fi eliminat doar din coadă (de la sfârșit).*

că obiectul poate fi îndepărtat (eliminat) pe măsură ce este adăugat ultimul. ■ *Push adaugă elemente în stivă și pop elimină din stivă elementele adăugate recent.* ■ *Într-o stivă se folosește un singur indicator (pointer).*	◆ *Enqueue adaugă elemente în partea din spate și Dequeue elimină elementele din partea din față a cozii.* ◆ *Într-o coadă simplă sunt folosiți doi indicatori (pointeri).*

4. Elaborarea unui fișier antet stiva.h și coada.h în limbajul C++

Model de fișier antet stiva.h

```
#include <iostream>
using namespace std;
#define MAX 100
struct stiva{
        int *s; // elemente stiva
        int v;  // varf stiva
};
stiva creaza_stiva (void){
        stiva rezultat; rezultat.s=new int [MAX];
        rezultat.v=-1; return rezultat;
}
int este_plina (struct stiva st){
        return (st.v==MAX-1)?1:0;
}
int este_goala (struct stiva st){
        return (st.v==-1)?1:0;
}
int push (struct stiva *st,int elem){ // inserare
// parametrul st este dat cu * pentru ca se modifica valoarea lui
(valoarea trebuie sa fie vizibila in exterior)
        if (!este_plina(*st))  {
                st->v++; st->s[st->v]=elem; return 0;
        }
        else return -1;
}
int pop (struct stiva *st){ // eliminare
// parametrul st este dat cu * pentru ca se modifica valoarea lui
(valoarea trebuie sa fie vizibila in exterior)
        if (!este_goala(*st))  {
```

```
                int rezultat; rezultat=st->s[st->v];
                st->v--; return rezultat;
        }
        else return -1;
}
int top (struct stiva st){ // varful stivei
// parametrul st este dat cu * pentru ca se modifica valoarea lui
(valoarea trebuie sa fie vizibila in exterior)
        if (!este_goala(st)) return st.s[st.v];
        return -1;
}
void print (struct stiva st){
        cout<<"stiva: ";
        if (!este_goala(st))   {
                int k;
                for (k=st.v;k>=0;k--) cout<<st.s[k]<<"->";
        } cout<<endl;
}
void distruge_stiva (struct stiva *rezultat){
        delete rezultat→s; rezultat->v=-1;
}
```

Implementarea bibliotecii stiva.h

```
#include "stiva.h"
int main (int argc,char **argv){
        struct stiva st=creaza_stiva();
        push(&st,1); // parametrul st se transmite prin referinta
        push(&st,2); // parametrul st se transmite prin referinta
        push(&st,3); // parametrul st se transmite prin referinta
        print(st);
        cout<<"top()="<<top(st)<<endl;
        cout<<"pop()="<<pop(&st)<<endl; // parametrul st se trans-
mite prin referinta
        cout<<"pop()="<<pop(&st)<<endl; // parametrul st se trans-
mite prin referinta
        cout<<"pop()="<<pop(&st)<<endl; // parametrul st se trans-
mite prin referinta
        print(st); distruge_stiva(&st); // parametrul st se trans-
mite prin referinta
}
```

Model de fişier antet coada.h

```
#include <iostream>
using namespace std;
#define MAX 100
struct coada{
        int *c; // elemente stiva
        int p,u;// pozitie primul element, pozitie ultimul element
};
struct coada creaza_coada (void){
        struct coada rezultat; rezultat.c=new int [MAX];
         rezultat.p=rezultat.u=-1; return rezultat;
}
int este_plina (struct coada cd){
        return (cd.p==0 && cd.u==MAX-1)?1:0;
}
int este_goala (struct coada cd){
        return (cd.p==-1 && cd.u==-1)?1:0;
}
int ins (struct coada *cd,int elem){
        if (!este_plina(*cd))  {
                if (este_goala(*cd)) cd->p++;
                cd->u++; cd->c[cd->u]=elem; return 0;
        } else return -1;
}
int del (struct coada *cd){
        if (!este_goala(*cd))  {
                int rezultat; rezultat=cd->c[cd->p]; int k;
                for (k=cd->p;k<cd->u;k++)
                        cd->c[k]=cd->c[k+1]; cd->u--;
                if (cd->u==-1) cd->p=-1;
                return rezultat;
        } else return -1;
}
int first (struct coada cd){
        if (cd.p!=-1) return cd.c[cd.p]; return -1;
}
int last (struct coada cd){
        if (cd.u!=-1) return cd.c[cd.u]; return -1;
}
void print (struct coada cd){
        cout<<"coada : ";
        if (!este_goala(cd)){
                int k;
```

```
                for (k=cd.p;k<=cd.u;k++) cout<<cd.c[k]<<"<-";
        } cout<<endl;
}
void distruge_coada (struct coada *rezultat){
        delete rezultat->c; rezultat->p=rezultat->u=-1;
}
```

Implementarea bibliotecii coada.h

```
#include "coada.h"
int main (int argc,char **argv){
        struct coada cd=creaza_coada();
        ins(&cd,1); // parametrul cd se transmite prin referinta
        ins(&cd,2); // parametrul cd se transmite prin referinta
        ins(&cd,3); // parametrul cd se transmite prin referinta
        print(cd); cout<<"first()="<<first(cd)<<endl;
        cout<<"last()="<<last(cd)<<endl;
        cout<<"pop()="<<del(&cd)<<endl; // parametrul cd se
transmite prin referinta
        cout<<"pop()="<<del(&cd)<<endl; // parametrul cd se
transmite prin referinta
        cout<<"pop()="<<del(&cd)<<endl; // parametrul cd se
transmite prin referinta
        print(cd);
        distruge_coada(&cd); // parametrul cd se transmite prin
referinta
}
```

Rezultatul execuţiei:

stiva.h

```
stiva : 3->2->1->
top()=3
pop()=3
pop()=2
pop()=1
stiva :
```

Coada.h

```
coada : 1<-2<-3<-
first()=1
last()=3
pop()=1
pop()=2
pop()=3
coada :
```

5. Răspunsuri la testele grilă

	Testul 1 *pag. 60*	Testul 2 *pag. 90*	Testul 3 *pag. 122*	Testul 4 *pag. 148*	Testul 5 *pag. 173*	Testul 6 *pag. 204*
1	*adevărat*	*adevărat*	*adevărat*	*adevărat*	*adevărat*	*fals*
2	*fals*	*fals*	*fals*	*adevărat*	*adevărat*	*adevărat*
3	*adevărat*	*1 2 3 5 7*	*adevărat*	*fals*	*fals*	*adevărat*
4	*căutare*	*7 3*	*LCDÎ*	*3 soluții*	*3 soluții*	*fals*
5	*omogene*	*2/8 5/10*	*LCSÎ*	*3 soluții*	*3 soluții*	*adevărat*
6	*struct*	*5 11 32 4*	*3 soluții*	*peek()*	*3 soluții*	*O (log 2 N)*
7	*2 soluții*	*2 soluții*	*nod*	*abstractă*	*push()*	*1 2 4 5 3 6 7 8*
8	*3 soluții*	*3 soluții*	*cap*	*realloc()*	*peek()*	*S-D-V*
9	*3 soluții*	*3 soluții*	*front*	*3 soluții*	*swap()*	*4*
10	*sfârșit*		*end*	*5 4 1*	*dequeue*	*postordine*
11			*splice*	*s=[1] & el=4*	*3 4 5*	*abc+*ead+/-h+*
12			*remove*		*8*	*ab+ac+*bc+**
13						*6 2 5*
14						*3 4 0 2 $*

NOTIȚE