

Capitolul 5. Structuri de date alocate dinamic

Structurile de date alocate dinamic sunt structuri ale căror componente se alocă în cursul execuției programului (dinamic).

Există mai multe tipuri de structuri dinamice: *liniare*, *arborescente* și *rețea*. Structurile liniare se mai numesc și *liste*.

Pentru alocarea dinamică a acestor structuri se utilizează variabilele dinamice.

Variabile dinamice (pointeri)

Prin *variabilă dinamică* (pointer) se înțelege o variabilă care reține la un moment dat adresa de memorie a unei alte variabile.

Declararea unei variabile pointer se face astfel:

Pascal	C/C++
var <nume_variabilă> : ^<tip>;	<tip> * <nume_variabilă>;

Se mai spune că ^<tip> (în Pascal), respectiv, <tip>* (în C/C++) reprezintă un nou tip de date, numit *tipul pointer*.

De exemplu, declararea unei variabile **p** de tip pointer care reține adresa unei alte variabile de tip întreg, se face astfel:

Pascal	C/C++
var p : ^Integer;	int *p;

Pointerii sunt date care se memorează într-o zonă specială din memoria internă, numită *heap*.

În Pascal este definită constanta **NIL** iar în C/C++ constanta **NULL** având ca valoare pointerul care nu conține nici o adresă, adică *pointerul nul*.

Ca operatori specifici privind pointerii amintim:

- *operatorul de referențiere* (@ în Pascal, respectiv, & în C/C++), astfel că @x în Pascal, respectiv, &x în C/C++ conține adresa variabilei statice x;
- *operatorul de adresare* (^ în Pascal, respectiv, * în C/C++), astfel că p^ în Pascal, respectiv, *p în C/C++ conține valoarea pe care o adresează pointerul p.

Iată un exemplu de utilizare a variabilelor dinamice și a operatorilor specifici:

Pascal	C/C++
<pre>var x: Integer; p: ^Integer; Begin x:=10; p:=@x; WriteLn; Write('p^ este chiar x: ', p^); End.</pre>	<pre>#include <iostream.h> void main(void) { int x=10, *p; cout<<"\nVar. x e la adr: "<<&x; cout<<" si are valoarea: "<<x; cout<<"\nVar. p are val.: "<<p; p=&x; cout<<" adresand obiectul: "<<*p; *p=20; cout<<"\nSi acum x are val.: "<<x; }</pre>

În continuare utilizăm câteva funcții specifice tipului pointer (cu precizarea că în C++ sunt operatori și nu funcții):

- funcția de alocare a spațiului de memorie pentru o variabilă pointer **p**

Pascal	C	C++
new(p);	(<tip*)>malloc(<tip> // <tip> este tipul variabilei spre care indică pointerul p	p= new <tip>;

- funcția de eliberare a spațiului de memorie ocupat de o variabilă pointer **p**

Pascal	C	C++
<code>dispose(p);</code>	<code>free(p);</code>	<code>delete p;</code>

- funcția ce verifică dacă există spațiu disponibil în HEAP pentru a fi alocat variabilei pointer **p**

Pascal	C/C++
<code>maxavail();</code>	<code>void malloc(unsigned nr_octeți);</code>

Funcția **maxavail** din Pascal întoarce dimensiunea celei mai mari zone de memorie din heap, iar funcția **malloc** din C/C++ întoarce un pointer care conține adresa primului octet al zonei alocate, însă, dacă spațiul disponibil este insuficient, funcția întoarce valoarea **NULL** (=0). Exemplu:

Pascal	C/C++
<pre>if sizeof(int) <= maxavail() then p = new() else WriteLn("Zona HEAP plina!");</pre>	<pre>if (malloc(sizeof(int)) p = new int; else cout << "Zona HEAP plina!";</pre>

Pentru ușurința expunerii, în continuare nu vom face verificări de alocare de spațiu în heap (dar vom utiliza funcțiile și operatorii atât pentru C cât și pentru C++).

5. 1 Liste simplu înlănțuite (definiție, operații specifice descrise în limbaj de programare)

Definiție

Prin *listă* (sau *listă liniară*) înțelegem un șir finit (eventual vid) de *elemente* (numite și *noduri* sau *componente*) aparținând unei mulțimi date $E = \{e_1, e_2, \dots, e_n\}$ care au următoarele proprietăți:

- există o relație de ordine între elementele listei în așa fel încât orice element e_i are ca *predecesor* pe e_{i-1} și ca *successor* pe e_{i+1} ;
- există un element în listă, și anume e_1 (primul element), care nu are nici un predecesor și care este numit *capul listei* sau *baza listei*;
- există un element în listă, și anume e_n (ultimul element), care nu are succesori și care este numit *element terminal al listei* sau *vârful listei*.

Prin urmare, lista este o structură de date între ale cărei componente există o *relație de ordine* (fiecare element al listei are un *singur succesori*, cu excepția ultimului element din listă care nu are nici un succesori și un *singur predecesor*, cu excepția primului element din listă care nu are nici un predecesor).

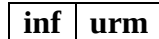
Exemple de liste: elementele unui vector, șirul format din numele din cartea de telefon (lista abonaților), elementele unui tablou oarecare pentru care există o ordine de memorare etc.

Componentele (elementele) unei liste pot fi date elementare (întregi, caractere, etc.) sau date structurate.

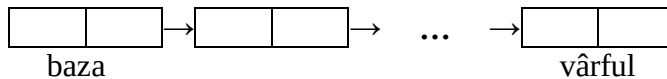
Memorarea listelor se poate face în mai multe moduri, în continuare prezentându-se două dintre ele.

Alocarea *secvențială* constă în a memora elementele listei în locații succesive de memorie, conform ordinei acestor elemente în listă și în același timp se memorează adresa de bază (adresa primului element al listei). Un exemplu tipic este cel în care elementele listei sunt memorate într-un vector, adresa de bază fiind adresa primului element al vectorului.

Alocarea *înlănțuită* presupune că fiecare element al listei este înlocuit cu o celulă formată din două părți: *o parte de informație* corespunzătoare elementului și *o parte de legătură* ce conține adresa celulei corespunzătoare următorului element (adresă numită obișnuit *pointer*). Ca și la alocarea secvențială, mai trebuie memorată o adresă de bază și, în plus, partea de legătură a ultimei celule primește o valoare ce nu poate desemna o legătură (o valoare specială, numită **NIL** în Pascal, respectiv **NULL** în C/C++ ce nu reprezintă nici o adresă de memorie). Celulele vor fi reprezentate sub forma



cu semnificații evidente. Legăturile vor fi precizate cu ajutorul unor săgeți, ca în figura următoare:



O listă definită astfel mai poartă numele de *listă liniară simplă* sau *listă liniară asimetrică*, deoarece parcurgerea ei nu se poate face decât într-un singur sens, plecând de la capul listei spre vârful ei.

Pentru simplificarea expunerii, presupunem în continuare că informația este o dată de tip întreg. În acest caz, implementarea unui nod al listei liniare simple se face astfel:

Pascal	C/C++
<pre> type pnod = ^nod; nod = record info :Integer; urm: pnod; end;</pre>	<pre> struct nod { int inf; nod *urm; } typedef struct nod NOD;</pre>

Operațiile cele mai importante care se efectuează cu listele sunt:

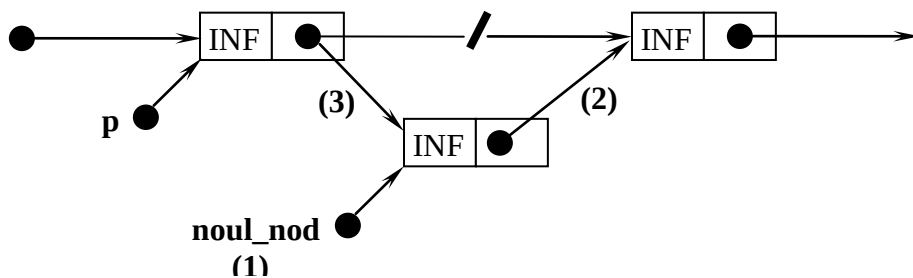
- crearea unei liste;
- parcurgerea și prelucrarea unei liste;
- adăugarea unui nou element în listă;
- ștergerea unui element din listă;
- întoarcerea numărului de elemente ale listei (cardinalul său).

La *adăugarea* unui nou nod în listă sunt posibile următoarele situații:

1. lista este vidă și se adaugă primul ei element:
2. lista nu e vidă, caz în care nodul se poate adăuga:
 - a. înaintea nodului indicat de pointerul **p**, fiind posibile două situații:
 - i. **p** indică primul element al listei;
 - ii. **p** nu indică primul element al listei;
 - b. după nodul indicat de pointerul **p**, fiind posibile două situații:
 - i. **p** indică ultimul element din listă;
 - ii. **p** nu indică ultimul element din listă.

Pentru a evita aceste situații, se poate recurge la un mic artificiu creând lista de la început cu două noduri fictive (în care nu se memorează nici o informație) și astfel nu mai apar atâtea cazuri pentru adăugarea unui nou nod. Cele două noduri fictive se mai numesc *santinele*. În acest fel, la adăugarea unui nou nod, nu mai este necesară verificarea cazurilor particulare în care lista e vidă sau în care dorim să adăugăm un nou nod înaintea primului nod din listă sau după ultimul nod din listă.

Adăugarea unui nou element după un nod oarecare din listă indicat de pointerul **p** se face astfel:



1. se alocă zona de memorie pentru **noul_nod**;
2. se copiază în **urm** din **noul_nod** adresa **urm** din nodul indicat de **p** (adică adresa nodului următor lui **p**, deci se face legătura cu succesorul noului nod);
3. se memorează adresa **noul_nod** în **urm** din **p** (deci legătura predecesorului noului nod)
4. se memorează informația în câmpul **inf** din **noul_nod**.

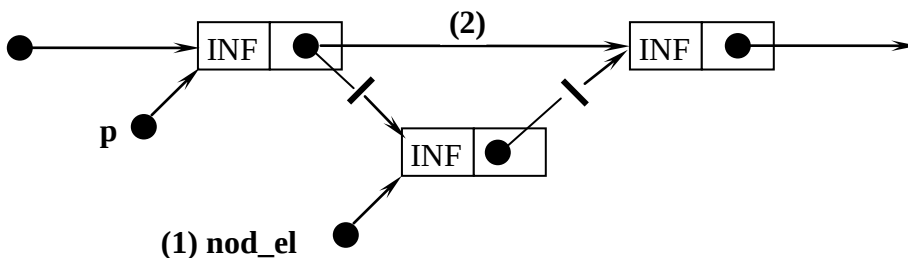
Variantă Pascal	Variantă C/C++
<code>new(noul_nod);</code> {1}	<code>noul_nod=new NOD;</code> // 1
<code>noul_nod^.urm:=p^.urm;</code> {2}	<code>noul_nod->urm=p->urm;</code> // 2
<code>p^.urm:=noul_nod;</code> {3}	<code>p->urm=noul_nod;</code> // 3
<code>ReadLn(noul_nod^.inf);</code> {4}	<code>cin>>noul_nod->inf;</code> // 4

Facem observația că trebuie respectată cu strictețe ordinea operațiilor deoarece, de exemplu, dacă s-ar efectua operația 3 înaintea operației 2, s-ar pierde adresa nodului care urma inițial după **p** (în mod iremediabil). Doar operația 4 poate fi efectuată oriunde după operația 1 (adică, oricând, după ce s-a alocat spațiu pentru noul nod).

Într-o listă simplu înălțuită putem adăuga un nou nod și *înaintea* unui nod indicat de un pointer **p**, însă printr-un mic artificiu. De fapt creăm noul nod tot după cel indicat de **p**, mutăm în el informația din nodul indicat de **p** (aflat în fața noului nod) după care noua informație o depunem în nodul din față, momentan indicat de **p** (după care **p** va memora adresa nodului introdus în listă).

Variantă Pascal	Variantă C/C++
<code>new(noul_nod);</code> {1}	<code>noul_nod=new NOD;</code> // 1
<code>noul_nod^.urm:=p^.urm;</code> {2}	<code>noul_nod->urm=p->urm;</code> // 2
<code>p^.urm:=noul_nod;</code> {3}	<code>p->urm=noul_nod;</code> // 3
<code>noul_nod^.inf:=p^.inf;</code> {4}	<code>noul_nod->inf=p->inf;</code> // 4
<code>ReadLn(p^.inf);</code> {5}	<code>cin>>p->inf;</code> // 5
<code>p:=noul_nod;</code> {6}	<code>p=noul_nod;</code> // 6

Ștergerea unui element în listă, aflat după un nod indicat de pointerul **p** se face astfel:



1. se memorează în pointerul **nod_el** adresa succesorului lui **p** (adică adresa nodului ce urmează a fi eliminat);

2. se copiază în **urm** din nodul indicat de **p** adresa nodului ce urmează după nodul ce se va elimina (adică se face legătura cu succesorul nodului ce se va elimina);
3. se eliberează zona de memorie ocupată de nodul ce se elimină.

Pascal	C	C++
nod_el:=p^.urm; {1}	nod_el=p->urm; //1	nod_el=p->urm; // 1
p^.urm:=p^.urm^.urm; {2}	p->urm=p->urm->urm; //2	p->urm=p->urm->urm; //2
dispose(nod_el); {3}	free(nod_el); //3	delete nod_el; // 3

Ca și la adăugarea unui nou nod în listă, și la *ștergerea* unui nod din listă sunt posibile mai multe situații. Listele create cu *santinele* elimină verificarea unor cazuri particulare ca, de exemplu, cel în care dorim să eliminăm ultimul nod din listă sau cel în care dorim eliminarea unui nod aflat după cel indicat de **p** și pointerul **p** îl indică chiar pe ultimul (deci, după el nu mai există nici un nod care sa poată fi eliminat).

Un caz particular este și eliminarea nodului din capul listei:

Pascal	C	C++
nod_el:=cap; {1}	nod_el=cap; // 1	nod_el=cap; // 1
cap:=cap^.urm; {2}	cap=cap->urm; // 2	cap=cap->urm; // 2
dispose(nod_el); {3}	free(nod_el) // 3	delete nod_el; // 3

Implementarea creării listei liniare simplu înlănțuite se face astfel:

Pascal	C/C++
<pre> Program creare; type pnod = ^nod; nod = record inf :Integer; urm: pnod; end; var cap, p,noulnod: pnod; n, i: Integer; Begin readln(n); new(p); cap:=p; readln(p^.inf); p^.urm:=NIL; for i:=2 to n do begin new(noulnod); readln(noulnod^.inf); p^.urm:=noulnod; p:=noulnod; end; p^.urm:=NIL; End. </pre>	<pre> #include <iostream.h> struct nod { int inf; nod *urm; }; typedef struct nod NOD; NOD *cap, *p, *noulnod; int n, i; void main() { cin>>n; p=new NOD; cap=p; cin>>p->inf; p->urm=NULL; for (i=2;i<=n;i++) { noulnod=new NOD; cin>>noulnod->inf; p->urm=noulnod; p=noulnod; }; noulnod->urm=NULL; } </pre>

Pentru a *parcurge* lista creată, utilizăm o structură repetitivă astfel:

Varianta Pascal	Varianta C/C++
<pre> p=cap; while (p<>NIL) do begin Write(p->inf, " "); p=p^.urm End </pre>	<pre> p=cap; while (p!=NULL) { cout<<p->inf<<" "; p=p->urm; } </pre>

adică inițializăm pointerul **p** cu adresa capului listei, după care, în mod repetat, afișăm informația (sau o prelucrăm, în funcție de necesități) iar prin instrucțiunea **p:=p^.urm** în **Pascal**, respectiv **p=p->urm** în **C/C++**, ne poziționăm pe următoarea înregistrare până când ajungem la ultimul nod unde următoarea adresă este **NIL** în **Pascal**, respectiv

NULL în **C/C++**. Adăugați aceste instrucțiuni la sfârșitul programului anterior pentru a afișa lista creată.

Este evident că alocarea înlănțuită necesită mai multă memorie decât cea secvențială. Trebuie menționat însă că în aplicații partea de informație este mult mai mare decât cea de legătură. Pe de altă parte, alocarea înlănțuită elimină necesitatea deplasării de elemente pentru operațiile de introducere și de scoatere de elemente din listă. Un alt avantaj al alocării înlănțuite este acela că permite folosirea unui spațiu de memorare comun pentru mai multe liste, ceea ce implică o economie de memorie, având în vedere numeroasele operații de introducere și de scoatere a elementelor din listă (liste).

În cazul în care se impun anumite restricții asupra operațiilor de inserare, ștergere sau consultare a elementelor unei liste liniare, se obțin câteva cazuri particulare de liste mult folosite în practică: stiva, coada și lista cu două capete.

Stiva

O listă liniară în care inserările, ștergerile și regăsirea elementelor se fac la un singur capăt al listei se numește *stivă*, *pilă*, *stack* sau *listă LIFO* (Last-IN, First-Out, adică ultimul intrat va fi primul ieșit).

Astfel de liste se utilizează mult în cadrul compilatoarelor și pentru scrierea de programe reentrante (programe ce nu sunt modificate de propria execuție). Stiva este utilizată la apelul subprogramelor și mai ales în recursivitate.

Coada

O listă liniară în care inserările elementelor se fac la un capăt al listei (sfârșitul listei), iar consultările și ștergerile de elemente se fac la celălalt capăt al listei (capul listei) se numește *coadă*, *fir de așteptare* sau *listă FIFO* (First-IN, First-Out, adică primul intrat va fi primul ieșit sau, mai bine spus, "primul venit - primul servit").

Astfel de liste se utilizează mult în sistemele de operare (în rutinele de alocare a resurselor) sau în programe de simulare.

Lista cu două capete

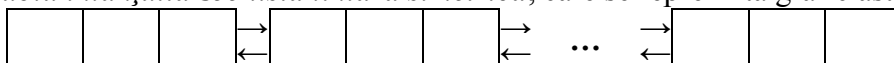
O listă liniară în care inserările, ștergerile și regăsirea elementelor se fac la ambele capete ale listei.

5. 2 Liste dublu înlănțuite (definiție, operații specifice descrise în limbaj de programare)

Dacă elementele unei liste liniare sunt formate din triplete de forma

ant	inf	urm
------------	------------	------------

unde **inf** reprezintă informația nodului, **ant** este adresa către predecesor (nodul anterior), iar **urm** este adresa către succesor (nodul următor), atunci se obține o *listă liniară dublu înlănțuită* sau *listă liniară simetrică*, care se reprezintă grafic astfel:

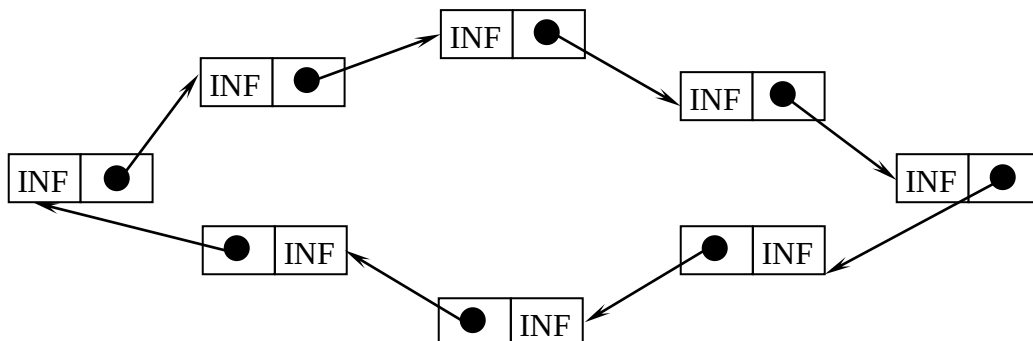


O astfel de listă poate fi parcursă în ambele sensuri, deci prezintă două grade de libertate.

Operațiile specifice listelor dublu înlănțuite se bazează pe aceeași logică așa cum am văzut la listele simplu înlănțuite, doar că trebuie să ținem seama că mai există și legătura spre elementul anterior și să dăm mare atenție ordinii în care realizăm legăturile pentru a nu pierde informații.

5. 3 Liste circulare (definiție, operații specifice descrise în limbaj de programare)

Dacă relația de ordine dintre elementele unei liste liniare este în așa fel definită încât ultimul element al listei are ca succesor primul element al listei, atunci se obține un nou tip de listă, și anume o *listă circulară* sau un *inel*. Reprezentarea grafică pentru o listă circulară simplu înlănțuită este următoarea:



O listă circulară poate fi *asimetrică* sau *simetrică* după cum elementele listei sunt dublete, respectiv triplete, adică conțin un pointer sau doi pointeri.

Listele circulare se mai numesc și *liste închise*, celelalte purtând numele de *liste deschise*.

La operațiile specifice listelor circulare trebuie să ținem cont, în plus, și de legăturile existente între ultimul nod și primul nod.

Atragem atenția asupra greșelilor frecvente în utilizarea alocării dinamice pentru liste:

- nerespectarea succesiunii corecte a operațiilor;
- prelucrarea (citirea, atribuirea) informațiilor înaintea alocării dinamice;
- stabilirea incorectă a legăturilor (de exemplu, pentru listele circulare la capete se uită să se închidă cercul de legături, iar pentru cele necirculare se uită să se stabilească legătura nulă);
- în momentul ștergerii sau inserției unui nod nu se actualizează corect legăturile;
- pierderea poziției capului listei;
- ștergerea unui nod înaintea preluării (memorării) legăturii următoare;
- inserția unui nod înaintea găsirii nodului anterior.

5. 4 Teste grilă (de la Bac)

1. Structura de date la care se aplică principiul „primul venit, primul ieșit”: (first in, first out) este:
 - a. lista înlănțuită
 - b. stiva
 - c. Coadă
 - d. Graf orientat

Bacalaureat 2009 (varianta 25, II. 1)
2. Nodurile unei liste dublu înlănțuite rețin în câmpurile **info**, **adp** și **adu** o informație numerică, adresa nodului precedent și respectiv adresa nodului următor din listă. Știind că lista este corect construită și că două noduri **p** și **q** ale acesteia se învecinează, atunci:

Varianta Pascal

- a. $p^{.adp}=q^{.adu}$ b. $p^{.adu}=q^{.adu}$ c. $p^{.adp}=q$ d. $p^{.adp}=q^{.adp}$

Variantă C/C++

- a. $p \rightarrow adp == q \rightarrow adu$ b. $p \rightarrow adu == q \rightarrow adu$ c. $p \rightarrow adp == q$ d. $p \rightarrow adp == q \rightarrow adp$

Bacalaureat 2006 (Simulare - subiectul I.6)

3.

Variantă Pascal

Se consideră o listă simplu înlănțuită ale cărei noduri rețin în câmpul **urm** adresa nodului următor al listei sau **nil** dacă nu există un element următor. Pentru inserarea unui nod aflat la adresa **p** imediat după un nod al listei aflat la adresa **q** se utilizează unele dintre următoarele atribuiri: 1) $p^{.urm}=q$ 2) $q^{.urm}=p$ 3) $p:=q^{.urm}$ 4) $q:=p^{.urm}$ 5) $p^{.urm}=q^{.urm}$ 6) $q^{.urm}=p^{.urm}$

Stabiliți care dintre acestea se utilizează și în ce ordine:

- a. 3 6 b. 2 4 c. 5 2 d. 2 3

Variantă C/C++

Se consideră o listă simplu înlănțuită ale cărei noduri rețin în câmpul **urm** adresa nodului următor al listei sau **NULL** dacă nu există un element următor. Pentru inserarea unui nod aflat la adresa **p** imediat după un nod al listei aflat la adresa **q**, se utilizează unele dintre următoarele atribuiri: 1) $p \rightarrow urm = q$; 2) $q \rightarrow urm = p$; 3) $p = q \rightarrow urm$; 4) $q = p \rightarrow urm$; 5) $p \rightarrow urm = q \rightarrow urm$; 6) $q \rightarrow urm = p \rightarrow urm$; .

Stabiliți care dintre acestea se utilizează și în ce ordine:

- a. 3 6 b. 2 4 c. 5 2 d. 2 3

Bacalaureat 2006 (Simulare - subiectul I.7)

4.

Variantă Pascal

Într-o listă simplu înlănțuită, cu cel puțin patru elemente, fiecare element reține în câmpul **urm** adresa elementului următor din listă. Dacă **p**, **q** și **r** sunt adresele a trei elemente din listă astfel încât $p^{.urm}=q^{.urm}^{.urm}$ și $r^{.urm}=q$ atunci ordinea logică a elementelor în listă (elementele fiind identificate prin adrese) este:

- a. q, r, p b. p, r, q c. r, q, p d. P, q, r

Variantă C/C++

Într-o listă simplu înlănțuită, cu cel puțin patru elemente, fiecare element reține în câmpul **urm** adresa elementului următor din listă. Dacă **p**, **q** și **r** sunt adresele a trei elemente din listă astfel încât $p \rightarrow urm == q \rightarrow urm \rightarrow urm$ și $r \rightarrow urm == q$ atunci ordinea logică a elementelor în listă (elementele fiind identificate prin adrese) este:

- a. q, r, p b. p, r, q c. r, q, p d. P, q, r

Bacalaureat 2007 (Varianta 1. subiectul I.4)

5.

Într-o listă simplu înlănțuită, cu cel puțin patru elemente, fiecare element reține în câmpul **adr** adresa elementului următor din listă, iar **q** este adresa ultimului element din listă. Atunci **p** este adresa antepenultimului element din listă dacă și numai dacă este satisfăcută condiția:

Variantă Pascal

- a. $Q^{.adr}^{.adr}=p$ b. $p^{.adr}=q$
c. $P^{.adr}^{.adr}=q$ d. $q^{.adr}=p^{.adr}^{.adr}$

Variantă C/C++

- a. $q \rightarrow \text{adr} \rightarrow \text{adr} == p$
- c. $p \rightarrow \text{adr} \rightarrow \text{adr} == q$

- b. $p \rightarrow \text{adr} == q$
- d. $q \rightarrow \text{adr} == p \rightarrow \text{adr} \rightarrow \text{adr}$

Bacalaureat 2007 (Variantă 3, subiectul I.4)

6. Într-o listă simplu înlănțuită, cu cel puțin patru elemente, fiecare element reține în câmpul **urm** adresa elementului următor din listă, iar **p** memorează adresa celui de-al treilea element din listă. Atunci **q** reține adresa primului element din listă dacă și numai dacă este satisfăcută condiția:

Variantă Pascal

- a. $p.^{\text{urm}}.^{\text{urm}} = q.^{\text{urm}}$
- b. $p.^{\text{urm}}.^{\text{urm}} = q$
- c. $q.^{\text{urm}}.^{\text{urm}}.^{\text{urm}} = p.^{\text{urm}}$
- d. $q.^{\text{urm}}.^{\text{urm}} = p.^{\text{urm}}$

Variantă C/C++

- a. $p \rightarrow \text{urm} \rightarrow \text{urm} == q \rightarrow \text{urm}$
- b. $p \rightarrow \text{urm} \rightarrow \text{urm} == q$
- c. $q \rightarrow \text{urm} \rightarrow \text{urm} \rightarrow \text{urm} == p \rightarrow \text{urm}$
- d. $q \rightarrow \text{urm} \rightarrow \text{urm} == p \rightarrow \text{urm}$

Bacalaureat 2007 (Variantă 4, subiectul I.5)

7. Variantă Pascal

Într-o listă simplu înlănțuită, cu cel puțin două elemente, fiecare element reține în câmpul **urm** adresa elementului următor din listă, iar **q** memorează adresa penultimului element din listă. Dacă **p** reține adresa unui element ce urmează a fi adăugat la sfârșitul listei și $p.^{\text{urm}}$ are valoarea **nil**, stabiliți care din următoarele este o operație corectă de adăugare:

- a. $p.^{\text{urm}} := q$
- b. $q.^{\text{urm}} := p$
- c. $q.^{\text{urm}}.^{\text{urm}} := p$
- d. $p.^{\text{urm}}.^{\text{urm}} := q$

Variantă C/C++

Într-o listă simplu înlănțuită, cu cel puțin două elemente, fiecare element reține în câmpul **urm** adresa elementului următor din listă, iar **q** memorează adresa penultimului element din listă. Dacă **p** reține adresa unui element ce urmează a fi adăugat la sfârșitul listei și $p \rightarrow \text{urm}$ are valoarea **NULL**, stabiliți care din următoarele este o operație corectă de adăugare:

- a. $p \rightarrow \text{urm} = q$
- b. $q \rightarrow \text{urm} = p$
- c. $q \rightarrow \text{urm} \rightarrow \text{urm} = p$
- d. $p \rightarrow \text{urm} \rightarrow \text{urm} = q$

Bacalaureat 2007 (Variantă 5, subiectul I.7)

8. Într-o listă simplu înlănțuită, cu cel puțin trei elemente, fiecare element reține în câmpul **nr** un număr întreg și în câmpul **urm** adresa următorului element din listă. Dacă variabila **prim** reține adresa primului element din listă, stabiliți care dintre următoarele secvențe afișează suma tuturor numerelor memorate în listă mai puțin cele reținute de primul și ultimul element:

Variantă Pascal

- a. $s := 0; p := \text{prim};$
 $\text{while } (p.^{\text{urm}} \neq \text{nil}) \text{ do begin } p := p.^{\text{urm}}; s := s + p.^{\text{nr}} \text{ end};$
 $\text{write}(s)$
- b. $s := 0; p := \text{prim};$
 $\text{while } (p.^{\text{urm}} \neq \text{nil}) \text{ do begin } s := s + p.^{\text{nr}}; p := p.^{\text{urm}} \text{ end};$
 $\text{write}(s)$

- c. `s:=0; p:=prim^.urm;`
`while (p^.urm<>nil) do begin s:=s+p^.nr; p:=p^.urm end;`
`write(s)`
- d. `s:=0; p:=prim;`
`while (p^.urm<>nil) do begin p:=p^.urm; s:=s+p^.nr end;`
`write(s-p^.nr)`

Varianta C/C++

- a. `s=0; p=prim;`
`while (p->urm!=NULL) do { p=p->urm; s=s+p->nr ;}`
`cout<<s; / printf("%d",s);`
- b. `s=0; p=prim;`
`while (p->urm!=NULL) do {s=s+p->nr ; p=p->urm;}`
`cout<<s; / printf("%d",s);`
- c. `s=0; p=prim->urm;`
`while (p->urm!=NULL) do { s=s+p->nr ; p=p->urm;}`
`cout<<s; / printf("%d",s);`
- d. `s=0; p=prim;`
`while (p->urm!=NULL) do { p=p->urm; s=s+p->nr ; }`
`cout<<s-p->nr; / printf("%d",s-p->nr);`

Bacalaureat 2007 (Varianta 8, subiectul I.5)

9. Într-o listă circulară simplu înlănțuită alocată dinamic cu cel puțin un element, fiecare element reține în câmpul **nr** un număr întreg și în câmpul **urm** adresa următorului element din listă. Știind că variabila **p** reține adresa unui element din listă și variabila **t** este de același tip cu variabila **p**, stabiliți care dintre următoarele secvențe afișează toate valorile memorate în elementele listei, fiecare valoare fiind afișată exact o dată:

Varianta Pascal

- | | |
|---|---|
| <ul style="list-style-type: none"> a. <code>t:=p;</code>
 <code>while (t^.urm<>p) do begin</code>
 <code> write(t^.nr, ' '); t:=t^.urm</code>
 <code>end</code> c. <code>t:=p;</code>
 <code>while (t<>p) do begin</code>
 <code> write(t^.nr, ' '); t:=t^.urm</code>
 <code>end</code> | <ul style="list-style-type: none"> b. <code>t:=p;</code>
 <code>repeat</code>
 <code> write(t^.nr, ' '); t:=t^.urm</code>
 <code>until (t=p)</code> d. <code>t:=p^.urm;</code>
 <code>repeat</code>
 <code> write(t^.nr, ' '); t:=t^.urm</code>
 <code>until (t=p)</code> |
|---|---|

Varianta C/C++

- a. `t=p;`
`while(t->urm!=p){`
`cout<<t->nr<<" "; / printf("%d",t->nr);`
`t=t->urm; }`
- b. `t=p;`
`do{`
`cout<<t->nr<<" "; / printf("%d ",t->nr)`
`t=t->urm;`
`}while(t!=p) ;`
- c. `t=p;`

```

while(t!=p){
    cout<<t->nr<<" "; / printf("%d",t->nr);
    t=t->urm; }
d. t=p->urm;
do{
    cout<<t->nr<<" "; / printf <"%d", t->nr) ;
    t=t->urm;
}while{t!=p) ;

```

Bacalaureat 2007 (Varianta 9, subiectul I.1)

5. 5 Probleme rezolvate

1. Într-o listă liniară simplu înlănțuită, alocată dinamic, cu cel puțin 4 elemente, fiecare element reține în câmpul **urm** adresa elementului următor sau **NIL** (în Pascal), respectiv **NULL** (în C/C++) dacă nu există un element următor, iar în câmpul **info** o valoare întreagă. Știind că variabila **p** reține adresa primului element din listă, înlocuiți punctele de suspensie cu expresiile corespunzătoare, astfel încât secvența următoare să calculeze în variabila **s** suma tuturor valorilor elementelor listei.

Varianta Pascal

```

s:=.....;
while ..... do
begin
p:=p^.urm;
s:=s+p^.info
end;
write(s);

```

Varianta C/C++

```

s=...;
while ( ... )
{
p=p->urm;
s=s+p->info;
}
cout<<s; | printf("%d",s);

```

Bacalaureat 2009 (varianta 8, II. 4)

Rezolvare:

Varianta Pascal

```

s:=p^info;
while p^urm <> NIL do
begin
p:=p^.urm;
s:=s+p^.info
end;
write(s);

```

Varianta C/C++

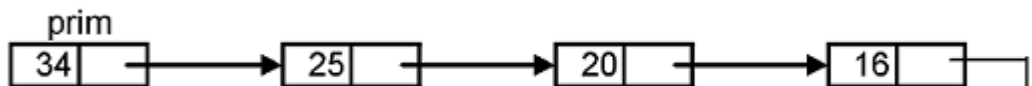
```

s=p->info;
while ( p->urm )
{
p=p->urm;
s=s+p->info;
}
cout<<s; | printf("%d",s);

```

2. O listă liniară simplu înlănțuită, alocată dinamic, reține în câmpul **info** al fiecărui element câte un număr natural din intervalul **[1,10000]**, iar în câmpul **adr**, adresa elementului următor din listă sau **NIL** (în Pascal), respectiv **NULL** (în C/C++), dacă nu există un element următor. Considerând că lista este creată și că adresa primului element este reținută în variabila **prim**, să se scrie declarațiile de tipuri și date necesare și secvența de program **PASCAL** (respectiv **C/C++**) care afișează pe ecran numerele memorate în listă, care sunt pătrate perfecte.

Exemplu: pentru lista



se vor afișa numerele 25 și 16.

Bacalaureat 2009 (varianta 85, II. 5)

Rezolvare. Parcurgem lista element cu element, vom verifica dacă elementul curent este pătrat perfect, în caz afirmativ îl vom afișa

Varianta PASCAL

```
type pnod=^nod;
    nod=record
        info:integer;
        adr:pnod;
    end;
var prim,q:pnod;
    k:integer;
.....
q:=prim;
while (q<>Nil) do
begin
    k:=q^.info;
    if sqrt(k)=int(sqrt(k))
        write(q^.info,' ');
    q:=q^.adr;
end;
```

Varianta C/C++

```
struct nod
{int info;
  nod * adr;
};
nod * prim, *q;
int k;
...
q=prim;
while(q)
{k= q->info;
  if(sqrt(k)==(int) sqrt(k))
      cout<<q->info<<' ';
  q=q->adr;
}
```

3. Să se scrie un program care să realizeze crearea, parcurgerea în sens direct și în sens invers și ștergerea unei liste simplu înlănțuite în mod recursiv (informația din noduri este un număr întreg).

Rezolvare. Vom aplica recursivitatea astfel:

- pentru creare este definită subprogramul recursivă **creare** care întoarce adresa nodului **cap** și nu are nici un parametru. Mai întâi se va alocă spațiu pentru nodul carent. Se va introduce informația acestuia și se va interoga utilizatorul dacă dorește introducerea altor noduri. Dacă răspunsul este afirmativ se continuă introducerea apelând recursiv subprogramul **creare**.

- pentru parcurgerea directă se va defini subprogramul **parcurgere** care are ca parametru adresa la nodul **cap** și nu întoarce nici o valoare. Mai întâi se va tipări informația nodului curent și apoi se vor parcurge celelalte noduri, apelând recursiv subprogramul **parcurgere** având ca parametru nodul următor.

- pentru parcurgerea inversă se definește subprogramul **parcursg_inv** care are ca parametru adresa nodului **cap** și nu întoarce nici o valoare. Se va parcurge mai întâi restul listei apelând în mod recursiv subprogramul **parcursg_inv** având ca parametru nodul următor și apoi se va tipări nodul curent, nemaifiind astfel nevoie de folosirea unui câmp care să conțină referința la nodul anterior. Observați cum schimbarea ordinii dintre tipărire și apel în cele două funcții furnizează cele două moduri de parcurgere: directă și inversă (iar recursivitatea ne ajută să parcurgem în sens invers chiar și liste simplu înlănțuite).

- pentru ștergerea listei se va defini subprogramul **ștergere**. Se va apela mai întâi ștergerea restului listei, apelând recursiv subprogramul **ștergere** și apoi se va elimina nodul curent.

Varianta Pascal

```
Program parcurgere_recursiva;
uses Crt;
type nod = ^adr;
    adr = record
        inf:Integer;
        leg:nod;
    end;
var cap: nod;
```

Varianta C/C++

```
#include <stdio.h>
#include <stdlib.h>
struct nod {int inf;
            struct nod* leg;};
struct nod* creare();
void parcurgere(struct nod*);
void parcursg_inv(struct nod*);
void stergere(struct nod*);
```

```

    c: Char;
function creare:nod;
var inf:Integer;
    p:nod;
begin
    WriteLn('Informatia: ');
    ReadLn(inf);
    if sizeof(nod)<=maxavail then
        begin
            new(p);
            p^.inf:=inf;
            WriteLn('Continuati
introducerea? (y/n) ');
            flush(input); c:=ReadKey;
            if c<>'n' then
                p^.leg:=creare
            else p^.leg:=NIL;
            creare:=p
        end
    else
        WriteLn('Depasire heap!')
    end;
procedure parcurgere(var p:nod);
begin
    if(p<>NIL) then begin
        Write(p^.inf, ' ');
        parcurgere(p^.leg)
    end
end;
procedure parcurg_inv(var p: nod);
begin
    if (p<>NIL) then
        begin
            parcurg_inv(p^.leg);
            Write(p^.inf, ' ');
        end
end;
procedure stergere(var p:nod);
begin
    if (p<>NIL) then
        begin stergere(p^.leg);
        dispose(p); end
end;
Begin
    cap:=creare;
    WriteLn('Parcurgere cap-coada:');
    parcurgere(cap); WriteLn;
    WriteLn('Parcurgere inversa:');
    parcurg_inv(cap);
    stergere(cap);
End.

```

```

void main()
{struct nod* cap;
  cap=creare();
  puts("Parcurgere cap-coada ");
  parcurgere(cap);
  puts("Parcurgere inversa: ");
  parcurgere_inv(cap);
  stergere (cap);
}
struct nod *creare()
{int inf;
 struct nod *p;
 printf("Informatia: ");
 scanf("%d",&inf);
 p= (struct nod*)
 malloc(sizeof(struct nod));
 p->inf=inf; fflush(stdin);
 printf("Continuati introducerea?
(y/n) ");
 if (getchar()=='y')
   p->leg=creare();
   else p->leg=NULL;
 return(p);
}
void parcurgere(struct nod *p)
{ if(p!=NULL) {
    printf("%d ", p->inf);
    parcurgere(p->leg);}
}
void parcurgere_inv(struct nod *p)
{ if (p!=NULL) {
    parcurgere_inv(p->leg);
    printf("%d ", p->inf);}
}
void stergere(struct nod *p)
{
  if (p!=NULL){ stergere(p->leg);
    free(p); }
}

```

4. Să se efectueze suma și produsul a două polinoame folosind liste simplu înlanțuite.

Rezolvare. Pentru aceasta vom face crearea unei liste corespunzătoare coeficienților unui polinom. Lista va avea ca informație gradul și coeficientul fiecărui termen de

coeficient nenul. Vom defini o funcție **canonic** care va elimina nodurile redundante din lista de termeni ai unui polinom prin păstrarea fiecărui grad o singură dată: dacă există două noduri cu același grad, unul din ele va fi eliminat iar coeficientul celuilalt va lua valoarea sumei coeficienților celor doi termeni.

Pentru a calcula suma a două polinoame este suficient să concatenăm listele celor două polinoame într-o a treia listă și să apelăm subprogramul **canonic** pentru această listă.

Calculul produsului se va face prin procesarea tuturor perechilor de termeni (unul din fiecare polinom) astfel:

- fiecare pereche va genera un nod în polinomul rezultat;
- gradul noului nod va fi egal cu suma gradelor nodurilor din pereche;
- coeficientul noului nod va fi egal cu produsul coeficienților termenilor din pereche.

După parcurgerea tuturor perechilor se va apela subprogramul **canonic**.

Varianta Pascal

```

Program
Suma_si_produs_polinoame;
Uses Crt;
type nod = ^adr;
      adr= record
          gr, cf: Integer;
          urm: nod;
      end;
var cap1, cap2, cap: nod;
    i: Integer;
function creare: nod;
var cap, p, q: nod;
    c: Char;
begin
    new(cap); { aici }
    Write; Write('Priumul coef. si
gr: ');
    ReadLn(cap^.cf, cap^.gr);
    cap^.urm:=NIL;
    p:=cap; c:='y';
    repeat
        WriteLn('Continuati?');
        flush(input); c:=ReadKey;
        if (c='n') then break;
        Write('Urm. coef. si gr: ');
        new(q); { aici }
        ReadLn(q^.cf, q^.gr);
        q^.urm:=NIL;
        p^.urm:=q; p:=q;
    until False;
    creare:=cap;
end;
procedure parcurgere(var p: nod);
var q: nod;
begin
    if (p=NIL) then
        begin
            WriteLn('Lista vida!');
        end
    else
        begin
            WriteLn('Lista vida!');
        end
end;

```

Varianta C/C++

```

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

typedef struct nod
{
    int gr; int cf;
    struct nod* urm;
} NOD;

NOD * cap1, *cap2, *cap;
int i;
void parcurgere(NOD* p);
NOD* creare(void);
NOD* suma (NOD* cap1, NOD*
cap2) ;
NOD* produs (NOD* cap1, NOD*
cap2);

NOD *create(void)
{ NOD *cap, *p, *q;

cap=(NOD*)malloc(sizeof(NOD));
printf("\nPriumul coef. si
gr: ");
scanf (" %d %d", &cap->cf,
&cap->gr);
cap->urm=NULL;
p=cap;
do { printf("Continuati?" );
    if (getch()=='n') break;
    printf("\nUrm. coef. si
gr: ");
    q=(NOD*)malloc(sizeof(NOD));
    scanf("%d %d", &q->cf, &q-
>gr);
    q->urm=NULL;
    p->urm=q; p=q;
}
while(1);

```

```

    end;
q:=p;
while q<>NIL do
begin
    Write(q^.cf,'x^', q^.gr,'+');
    q:=q^.urm;
end;
WriteLn(#8, ' ');
end;
procedure canonic(var cap:nod);
var p,q,r: nod;
begin
    p:=cap;
    while p<>NIL do
    begin
        q:=p^.urm;
        r:=p;
        while q<>NIL do
        begin
            if (p^.gr=q^.gr) then
            begin
                p^.cf:=p^.cf+q^.cf;
                r^.urm:=q^.urm; r:=q;
            end;
            dispose(q);
            end
            else r:=q;
            q:=q^.urm;
            end;
            p:=p^.urm;
            end;
            end;
            function suma(var
cap1,cap2:nod):nod;
var p,q, cap,r:nod;
begin
    if(cap1<>NIL) then new(cap);
    cap^.cf:=cap1^.cf;
cap^.gr:=cap1^.gr;
    cap^.urm:=NIL; r:=cap;
    p:=cap1^.urm;
    while p<>NIL do
    begin
        new(q);
        q^.cf:=p^.cf;
; q^.gr:=p^.gr; q^.urm:=NIL;
        r^.urm:=q; r:=q;
        p:=p^.urm;
    end;
    p:=cap2;
    while p<>NIL do
    begin
        new(q);
        q^.cf:=p^.cf;
q^.gr:=p^.gr; q^.urm:=NIL;
        if r<>NIL then r^.urm:=q;
        r:=q;

```

```

        return(cap);
    }
    void parcurgere(NOD *p)
    { NOD *q;
        if(!p){ printf("\nLista
vida!"); return;}
        for (q=p;q=q->urm)
            printf (" \n%2d x^%d+", q-
>cf, q->gr);
            printf (" \b " ) ;
        }
    void canonic(NOD *cap)
    { NOD *p,*q,*r;
        for(p=cap;p=p->urm)
            for(q=p->urm;q=r;q=q->urm)
                if (p->gr==q->gr)
                    { p->cf+=q->cf;
                        r->urm=q->urm; free(q);
                    }
        }
    NOD* suma (NOD* cap1, NOD* cap2)
    { NOD *p,*q,*cap,*r;
        if(cap1)
            cap=(NOD*)malloc(sizeof(NOD));
            cap->cf=cap1->cf; cap->
>gr=cap1->gr;
            cap->urm=NULL; r=cap;
            for (p=cap1->urm;p=p->urm)
                { q=(NOD*)malloc(sizeof(NOD));
                    q->cf=p->cf ;q->gr=p->gr;q->
>urm=NULL;
                    r->urm=q;r=q;
                }
                for (p=cap2;p=p->urm)
                    { q=(NOD*)malloc
(sizeof(NOD));
                        q->cf=p->cf; q->gr=p->gr;
                        q->urm=NULL;
                        if(r) r->urm=q;r=q;
                    }
                canonic(cap);
                return cap;
            }
        }
    NOD* produs (NOD* cap1, NOD*
cap2)
    { NOD *p,*q,*ult,*r,*cap; int
este_cap=0;
        este_cap=0;
        for (p=cap1;p=p->urm)
            for (q=cap2;q=q->urm)
                {if (!este_cap)
                    { este_cap=1;
                        cap=(NOD*)
malloc(sizeof(NOD));
                        cap->cf=p->cf*q->cf;

```

```

    p:=p^.urm
end;
canonic(cap);
suma:=cap;
end;
function produs (var
cap1,cap2:nod):nod;
var p,q,ult,r,cap:nod;
    este_cap:Boolean;
begin
    este_cap:=False;
    p:=cap1;
    while p<>NIL do
    begin
        q:=cap2;
        while q<>NIL do
        begin
            if (NOT este_cap) then
            begin
                este_cap:=True;
                new(cap);
                cap^.cf:=p^.cf*q^.cf;
                cap^.gr:=p^.gr+q^.gr;
                cap^.urm:=NIL;ult:=cap;
            end
            else begin
                new(r);
                r^.cf:=p^.cf*q^.cf;
                r^.gr:=p^.gr+q^.gr;
                r^.urm:=NIL;
                ult^.urm:=r;ult:=r;
            end;
            q:=q^.urm
        end;
        p:=p^.urm
    end;
    canonic(cap);
    produs:=cap
end;
Begin
    cap1:=create;canonic(cap1);
    WriteLn('Polinomul 1:');
    parcurgere(cap1);
    cap2:=create; canonic(cap2);
    WriteLn('Polinomul 2:');
    parcurgere(cap2);
    cap:=suma(cap1,cap2);
    WriteLn('Polinomul suma:');
    canonic(cap); parcurgere(cap);
    cap:=produs(cap1,cap2);
    WriteLn('Poiinomul produs:');
    canonic(cap);parcurgere(cap);
End.

```

```

    cap->gr=p->gr+q->gr;
    cap->urm=NULL;ult=cap;
    }
    else {
r=(NOD*)malloc(sizeof(NOD));
        r->cf=p->cf*q->cf;
        r->gr=p->gr+q->gr;
        r->urm=NULL;
        ult->urm=r;ult=r;
    }
    }
    canonic(cap);
    return cap;
}

void main()
{ clrscr();
    cap1=create();canonic(cap1);
    printf("\nPolinomul 1:" );
    parcurgere(cap1);
    cap2=create(); canonic(cap2);
    printf("\nPolinomul 2:");
    parcurgere(cap2);
    cap=suma(cap1,cap2);
    printf("\nPolinomul suma:");
    canonic(cap); parcurgere(cap);
    cap=produs(cap1,cap2);
    printf("\nPoiinomul produs:");
    canonic(cap);parcurgere(cap);
}

```


5. Să se creeze o listă dublu înălțuită cu nodurile memorând numere întregi, să se parcurgă atât în sens direct cât și-n sens invers, după care să se șteargă lista. Pentru fiecare operație se va scrie câte un subprogram.

Variantă Pascal

```
Type nod=^adr;
    adr=record
        inf:Integer;
        ant, urm:nod;
    end;
var nr:Integer;
    cap1, cap2: nod;

procedure creare(nr:Integer;
    var cap1, cap2:nod);
var i:Integer;
    p:nod;
begin
    new(cap1);
    Write('Informatia capului: ');
    ReadLn(cap1^.inf);
    cap1^.urm:=NIL;
    cap1^.ant:=NIL;
    cap2:=cap1;
    for i:=2 to nr do
    begin
        new(p);
        Write('Informatia nr.',i,' :');
        ReadLn(p^.inf);
        p^.ant:=cap2;cap2^.urm:=p;
        p^.urm:=NIL;cap2:=p
    end
end;
procedure parc(p:nod;tp:Integer);
begin
    if p<>NIL then
        if tp=1 then
            begin
                Write(p^.inf, ' ');
                parc(p^.urm, tp)
            end
        else
            begin
                Write(p^.inf, ' ');
                parc(p^.ant, tp)
            end
        end
end;
procedure sterg(p:nod);
begin
    if p<> NIL then
    begin
        sterg(p^.urm);
        dispose(p)
    end
end;
```

Variantă C/C++

```
#include <stdio.h>
#include <stdlib.h>
struct nod {int inf;
            struct nod *urm,*ant;
};
void creare(int nf,struct
nod**,struct nod**);
void parc(struct nod*,int tp);
void sterg(struct nod*);
void main()
{int nr;
    struct nod * cap1,*cap2;
    printf("Numarul de elemente: ");
    scanf("%d",&nr);
    creare (nr, &cap1, &cap2);
    puts("\nParcuregere directa");
    parc(cap1,1);
    puts("\nParcuregere inversa");
    parc(cap2,2);
    sterg(cap1);
}
void creare(int nr,struct nod
**cap1,
            struct nod **cap2)
{int inf, i;
    struct nod *p;
    *cap1=(struct nod*)
        malloc(sizeof(struct nod));
    printf ("Informatia capului: ");
    scanf (" %d", &inf);
    (*cap1)->inf=inf;
    (*cap1)->urm=(*cap1)->ant=NULL;
    *cap2=*cap1;
    for(i=2;i<=nr;i++)
    {p=(struct nod*)
        malloc(sizeof(struct nod));
        printf("Informatia nr.%d: ",i);
        scanf(" %d", &inf);
        p->inf=inf;
        p->ant=*cap2;
        (*cap2)->urm=p;
        p->urm=NULL;*cap2=p;
    }
}
void parc(struct nod *p,int tp)
{if(p)
    if(tp==1)
        {printf (" %d ", p->inf);
            parc(p->urm, tp);
        }
}
```

```

Begin
Write('Numarul de elemente: ');
ReadLn(nr);
creare (nr, cap1, cap2);
WriteLn('Parcurs directa');
parc(cap1,1); WriteLn;
WriteLn('Parcurs inversa');
parc(cap2,2);
sterg(cap1)
End.

```

```

else {printf ("%d ",p->inf);
      parc(p->ant, tp);
      }
}
void sterg(struct nod *p)
{if (p)
 {sterg(p->urm); free(p);}
}

```

6. Să se folosească o stivă alocată dinamic pentru a face conversia unui număr n din baza 10 într-o bază b mai mică decât 16 .

Rezolvare. Se memorează într-un vector resturile modulo **16** ale unui număr natural oarecare (adică toate cifrele posibile). Se vor depune în zona de informație utilă a fiecărui nod al stivei acel caracter care se găsește în poziția n modulo b . Apoi, traversând stiva și afișând resturile, obținem numărul în baza b .

Varianta Pascal

```

uses Crt;
type nr = ^adr;
      adr=record
          inf: Char;
          urm: nr;
      end;
const a: array[0..15] of Char=
      ('0','1','2','3','4','5',
       '6','7','8','9','A','B',
       'C','D','E','F');
var varf: nr; {varf stiva}
{stiva resturi modulo baza}
function cifre: nr;
var n, b: Integer;
    varf, p: nr;
begin
Write('Nr. de convertit: ');
ReadLn(n);
Write('Baza noua: '); ReadLn(b);
new(varf);
varf^.inf:=a[n mod b];
varf^.urm:=NIL;
n:=n div b;
while (n<>0) do
begin
new(p);
p^.inf:=a[n mod b];
p^.urm:=varf; varf:=p;
n:=n div b;
end;
cifre:=varf
end;
procedure scrie_cifre(p: nr);
var q: nr;
begin
q:=p;
while q<>NIL do

```

Varianta C/C++

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<iostream.h>
struct nod{int cod;
           struct nod *urm;};
typedef nod NOD;
NOD *prim,*ultim,*p;
void intrare()
{int codul;
printf("Intra clientul= ");
scanf("%d",&codul);
if(prim==NULL)
{prim=(NOD*)malloc(sizeof(NOD));
prim->cod=codul;
prim->urm=NULL;
ultim=prim;}
else
{p=(NOD*)malloc(sizeof(NOD));
p->cod=codul;
p->urm=NULL;
ultim->urm=p;
ultim=p;}
}
void iesire()
{if(prim==NULL)
{printf("Nu mai sunt clienti\n");
getch();}
else
{printf("Iese clientul %d \n",
prim->cod);
p=prim;
prim=prim->urm;
free(p);
}getch();
}
void afisare()

```

```

begin
  Write(q^.inf);
  q:=q^.urm
end
end;
Begin
  clrscr;
  varf:=cifre;
  scrie_cifre(varf);
  ReadLn
End.

```

```

{if (prim==NULL)
  printf("Nu e nimeni la coada\n");
else
{printf("Clientii de la coada:
\n");
  p=prim;
  do
  {printf("%d \n",p->cod);
   p=p->urm;}
  while (p!=NULL) ;
} getch();}
void main(void)
{char comanda;
  prim=NULL;
do {clrscr ();
  printf("Comanda: ");
  scanf("%c",&comanda);
  switch (comanda)
  {case 'i':intrare();break;
   case 'e' : iesire();break;
   case 'l':afisare();break;}}
  while(comanda!='s');
}

```

7. Se consideră un ghișeu la care se așteaptă la coadă pentru a fi serviți mai mulți clienți. Să se scrie un program care prelucrează comenzi de forma: **I** (intrarea la coadă), **E** (ieșirea de la coadă în urma servirii de la ghișeu), **L** (listarea persoanelor ce se află la coadă) și **S** (sfârșitul programului). Clienții sunt codificați numeric.

Rezolvare. Vom folosi o listă de tip coadă.

Varianta Pascal

```

uses Crt;
type nod=^adr;
      adr=record
          cod:Integer;
          urm:nod;
        end;
var prim,ultim,p:nod;
    comanda:Char;
procedure intrare;
var codul:Integer;
begin
  Write('Intra clientul=');
  ReadLn(codul);
  if prim=NIL then
  begin
    new(prim);
    prim^.cod:=codul;
    prim^.urm:=NIL;
    ultim:=prim end
  else
  begin
    new(p);
    p^.cod:=codul;
    p^.urm:=NIL;
    ultim^.urm:=p;
    ultim:=p
  end

```

Varianta C/C++

```

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<iostream.h>
struct nod{int cod_loc;
            struct nod *urm;};
typedef nod NOD;
NOD *prim,*ultim,*p;
void intrare()
{int codul;
  printf("Codul loc care intra=");
  scanf("%d",&codul);
  if(prim==NULL)
  //coada vida, se adauga primul el.
  {prim=(NOD*)malloc(sizeof(NOD));
   prim->cod_loc=codul;
   prim->urm=NULL;
   ultim=prim;}
  else
  {p=(NOD*)malloc(sizeof(NOD));
   p->cod_loc=codul;
   p->urm=NULL;
   ultim->urm=p;
   ultim=p;}
}
void iesire()//sterg primul el.

```

```

end;
end;
procedure iesire;
begin
    if prim=NULL then
        begin
            WriteLn('Nu mai sunt clienti');
            ReadLn end
        else
            begin
                Write('Iese clientul ',
                    prim^.cod);
                p:=prim;
                prim:=prim^.urm;
                dispose(p);
                ReadLn
            end;
            ReadLn
        end;
    procedure afisare;
    begin
        if prim=NULL then
            WriteLn('Nu e nimeni la coada')
        else
            begin
                Write('Clientii de la coada: ');
                p:=prim;
                repeat
                    Write(p^.cod, ' ');
                    p:=p^.urm
                until p=NULL
            end;
            ReadLn; ReadLn
        end;
    end;
    Begin
        prim:=NIL;
        repeat
            clrscr;
            Write('Comanda: ');
            Read(comanda);
            case comanda of
                'i':intrare;
                'e':iesire;
                'l':afisare
            end;
        until comanda='s'
    End.

```

```

{if(prim==NULL)
{printf("Depou gol\n");
getch();}
else
{printf("Iese locom. %d \n",
    prim->cod_loc);
p=prim;//salvez adr. primului
prim=prim->urm;
free(p);
}getch();
}
void afisare()
{if (prim==NULL)
    printf("Depoul este gol\n");
else
{printf("Locom. din depou: \n");
p=prim;
do
{printf("%d \n",p->cod_loc);
p=p->urm;}
while (p!=NULL) ;
} getch();}
void main(void)
{char comanda;
prim=NULL;
do {clrscr ();
printf("Comanda: ");
scanf("%c",&comanda);
switch (comanda)
{case 'i':intrare();break;
case 'e' : iesire();break;
case 'l':afisare();break;}}
while(comanda!='s');
}

```

5. 6 Probleme propuse

1. Să se scrie un subprogram care să realizeze inversarea legăturilor într-o listă liniară simplu înlănțuită.

Indicații. Pentru a avea sens inversarea, lista trebuie să aibă cel puțin două noduri. Vom utiliza doi pointeri **p** și **q** care rețin adresa a câte două noduri succesive și vom parcurge lista realizând inversarea legăturilor.

2. Să se realizeze următoarele funcții pentru: crearea unei liste cu cifrele unui număr pe care-l primește ca paramtru, adunarea, scăderea, înmulțirea și împărțirea numerelor memorate astfel în două liste.

3. Să se scrie un subprogram care citește pe rând cifrele unui număr întreg foarte mare. Să se utilizeze această funcție într-un program care citește numere foarte mari și realizează operațiile de bază cu ele (adunarea, scăderea, înmulțirea, împărțirea).

4. Fiind dată o listă simplu înlănțuită având ca informații numere reale, să se afișeze câte numere negative, câte numere egale cu 0 și câte numere pozitive conține lista.

5. Să se creeze o listă prin inserări succesive, astfel încât la fiecare pas lista să fie ordonată crescător.

Indicație. Pentru a crea lista ordonată crescător va trebui ca înainte de a insera o nouă valoare în listă să căutăm poziția sa corectă, apoi să inserăm valoarea pe poziția corectă, astfel încât la fiecare moment lista să fie sortată. Astfel, dacă lista este vidă sau valoarea care trebuie inserată este mai mică decât informația primului nod din listă, atunci inserarea se face la începutul listei. Altfel, vom lua un pointer **p** pe care-l deplasăm de la începutul listei spre vârful său până când nodul care urmează după cel indicat de **p** are o informație mai mare decât valoarea de inserat sau **p** este ultimul nod din listă, inserarea noii valori făcându-se după nodul indicat de **p**.

6. Un polinom cu coeficienți reali $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ poate fi memorat într-o listă simplu înlănțuită în care fiecare nod va reține coeficientul și gradul unui monom de forma $a_k x^k$ al polinomului. Definiți și creați structurile de date necesare, apoi scrieți câte un subprogram pentru:

a) calculul valorii unui polinom dat $P(x)$, pentru o valoare dată a argumentului x ;

b) afișarea polinomului ce reprezintă suma a două polinoame date, $P(x) + Q(x)$.

Indicații: Pentru a putea aduna într-o manieră foarte simplă două polinoame este recomandabil ca lista aferentă polinomului $P(x)$ să conțină **n** noduri, ce vor memora coeficienții și gradele monoamelor. Fie **coef** și **grad** cele două câmpuri ale fiecărui nod. Dacă un monom lipsește îi va reveni în listă un nod cu valoarea 0 în câmpul **coef**.

1) Fie **v** variabila în care vom calcula valoarea polinomului. Parcurgem lista într-un ciclu cu ajutorul unui pointer **q** și la fiecare pas adăugăm la **v** valoarea unui monom $a_k x^k$, care se poate exprima sub forma **coef*putere(x, grad)**. Funcția **putere(b,m)** returnează b^m .

2) Folosind reprezentarea indicată, parcurgem "în paralel" cele două liste cu ajutorul unor pointeri **q** și **r**. La fiecare pas al parcurgerii cei doi pointeri vor adresa noduri corespunzătoare unor monoame de același grad în cele două polinoame și însumăm coeficienții monoamelor corespunzătoare aceluiasi grad.

7. Fie două *polinoame rare* (în care majoritatea coeficienților sunt nuli). De exemplu, $P(x) = 5x^{16} + 3x^2 - 2$ este un polinom rar. Astfel de polinoame pot fi memorate ca o listă simplu înlănțuită în care sunt reținute monoamele în ordinea crescătoare a gradelor, pentru fiecare monom reținându-se în partea de informații coeficientul și gradul său. Spunem în acest caz că polinoamele sunt memorate în *forma condensată* (adică nu sunt memorate și monoamele cu coeficienți nuli, ceea ce ar duce la un consum inutil de memorie). Să se scrie un program care să realizeze suma celor două polinoame memorate în forma condensată.

8. Scrieți un subprogram care, primind ca parametri două liste liniare simplu înlănțuite de numere întregi, afișează reuniunea celor două liste, fără a folosi tablouri sau

alte structuri de date alocate static. *Reuniunea* a două liste este alcătuită din cheile lor comune și necomune, luate o singură dată.

9. Scrieți un subprogram care, primind ca parametri două liste liniare simplu înlanțuite de numere întregi, afișează intersecția celor două liste, fără a folosi tablouri sau alte structuri de date alocate static. *Intersecția* a două liste este alcătuită din cheile lor comune, luate o singură dată.

10. Scrieți un subprogram care, primind ca parametri două liste liniare simplu înlanțuite de numere întregi, afișează diferența celor două liste, fără a folosi tablouri sau alte structuri de date alocate static. *Diferența* a două liste este alcătuită din cheile primei liste care nu se află și-n a doua listă.

11. Scrieți un program care să creeze o listă liniară dublu înlanțuită în care nodurile conțin ca informație numere întregi și să realizeze principalele operații cu această listă (adăugarea unui nou nod, ștergerea unui nod, căutarea unui nod care conține ca informație o anumită valoare întreagă, etc.).

12. Scrieți un program care să creeze o listă liniară simplu înlanțuită circulară în care nodurile conțin ca informație numere întregi și să realizeze principalele operații cu această listă (adăugarea unui nou nod, ștergerea unui nod, căutarea unui nod care conține ca informație o anumită valoare întreagă, etc.).

13. Scrieți un program care să creeze o listă liniară dublu înlanțuită circulară în care nodurile conțin ca informație numere întregi și să realizeze principalele operații cu această listă (adăugarea unui nou nod, ștergerea unui nod, căutarea unui nod care conține ca informație o anumită valoare întreagă, etc.).

14. Scrieți un subprogram care, primind ca parametru o listă de numere naturale, afișează pătratele perfecte aflate în noduri cu număr de ordin impar (primul, al treilea, al cincilea, etc.) în lista dată.

Exemplu: pentru lista $L=(49,45,172,36,169)$, se vor afișa numerele 49 și 169.

15. Construiți o listă de numere naturale citite din fișierul "**nr.txt**" (în care fiecare nod va conține drept informație un număr din fișier), apoi scrieți un subprogram care afișează cheile listei a căror sumă a cifrelor este egală cu o valoare dată **s**. Numerele sunt scrise în fișier unul sub altul, fiecare pe câte un rând, și nu se cunoaște câte astfel de numere conține fișierul.

Exemplu: Pentru fișierul ce conține numerele 5, 123, 14, 85, 1121, 16 și $s=5$, se vor afișa numerele 5, 14 și 1121.

16. Se citește de la tastatură numere întregi, până la introducerea valorii **0** (care nu va face parte din șirul citit). Să se creeze o listă liniară simplu înlanțuită care să conțină drept chei numerele prime dintre cele citite. Nu se vor folosi vectori sau alte structuri de date alocate static.

17. Se citește de la tastatură un cuvânt de lungime maxim **80** de caractere. Memorând caracterele cuvântului într-o listă simplu înlanțuită (în care fiecare nod va conține în câmpul de informație un caracter), să se verifice dacă respectivul cuvânt este palindrom. Reamintim: un cuvânt este *palindrom* dacă, citind caracterele sale în ordine inversă, de la dreapta la stânga, obținem același cuvânt. De exemplu, cuvântul "cojoc" este palindrom.

18. Să se construiască o listă liniară simplu înlanțuită cu primele **n** numere naturale impare. Numerele se vor memora în ordine crescătoare în câmpurile de informație ale nodurilor, câte unul în fiecare nod.

Exemplu: pentru $n=8$, lista va fi (1->3->5->7->9->11->13->15).

19. Scrieți un subprogram care verifică dacă două liste liniare simplu înlanțuite sunt identice sau nu, returnând rezultatul logic al testării. Subprogramul va primi drept parametri adresele de început ale celor două liste. Spunem că două liste se consideră *identice*, dacă au același număr de noduri și nodurile cu același număr de ordine în cele două liste conțin aceeași informație.

20. Realizați un subprogram care concatenează două liste liniare simplu înlanțuite de numere întregi (prin *concatenarea* a două liste se înțelege "lipirea" celei de-a doua la sfârșitul primeia). Subprogramul va primi ca parametri pointerii către primul nod al celor două liste ce trebuie concatenate, și va returna un pointer către începutul listei rezultate prin concatenare.

21. Se citește de la tastatură un număr întreg cu maxim opt cifre. Să se afișeze oglinditul numărului folosind o listă simplu înlanțuită în nodurile căreia se memorează cifrele numărului. Prin *oglinditul* unui număr întreg înțelegem numărul obținut prin citirea cifrelor numărului dat în ordine inversă de la dreapta la stânga (palindrom numeric).

Exemplu: Dacă se citește numărul 742881, se va construi lista $L=(1->8->8->2->4->7)$.

22. Scrieți o funcție care, primind ca parametri un întreg **k** și un pointer **p** către primul nod al unei liste liniare simplu înlanțuite de numere întregi, șterge primele **k** noduri din listă, și returnează lista rămasă. Algoritmul se va baza pe ștergerea repetată a primului nod. Nu se vor folosi alte subprograme și nici structuri de date alocate static.

Exemplu: Pentru lista $L=(2->5->8->11->4->11->11->7)$ și $k=3$, după ștergere va rămâne lista $L2=(11->4->11->11->7)$.

23. Scrieți un program care, pentru o listă dată de n numere întregi, determină cea mai mare diferență în modul dintre două chei consecutive ale listei. Prima dintre cele două chei care alcătuiesc diferența localizată va fi mutată la începutul listei, iar a doua la sfârșitul listei.

Exemplu: Pentru lista $L=(-3->-1->8->2->12->9->14)$, va rezulta lista $L2=(2->-3->-1->8->9->14->12)$.

24. Scrieți un subprogram care, primind ca parametru un pointer **p** către începutul unei liste liniare simplu înlanțuite de numere întregi, crează alte două liste astfel: în prima listă se vor introduce numerele pozitive ale listei date, iar în cea de-a doua se vor memora numerele negative din lista dată.

25. Se citește un text caracter cu caracter din fișierul "**fraza.txt**". Textul este scris pe un singur rând în fișier, și poate conține orice caractere. Să se construiască o listă liniară dublu înlanțuită care să conțină caracterele distincte din text împreună cu frecvențele lor de apariție în cadrul textului, apoi să se afișeze conținutul listei pe ecran (pe fiecare rând se va scrie câte un caracter distinct și frecvența sa de apariție, separate printr-un spațiu).

26. Se dă un șir de numere întregi citite de la tastatură. Să se introducă aceste numere în ordine crescătoare într-o listă liniară simplu înlanțuită. Lista se va crea de la început ordonată, și nu se va aplica nici un algoritm de sortare.

Indicații: Introducem separat în listă primul număr. Pentru fiecare din restul numerelor trebuie să căutăm poziția pe care ar ocupa-o noul nod în listă, astfel încât numerele din noduri să fie tot timpul în ordine crescătoare:

- dacă numărul pe care-l introducem este mai mic decât cel din primul nod, atunci noul nod se va adăuga la începutul listei, modificându-se capătul acesteia;
- în caz contrar, parcurgem lista până la identificarea poziției pe care ar ocupa-o noul nod pentru ca lista să rămână sortată crescător după care inserăm nodul cu numărul citit și corectăm legăturile;
- mai există și posibilitatea ca numărul pe care-l adăugăm în listă să fie mai mare decât numărul din ultimul nod, situație în care adăugăm noul nod la sfârșitul listei.

27. Se citește de la tastatură n numere naturale, care se memorează într-o listă (fiecare nod va conține drept informație unul din cele n numere). Scrieți un program care, pentru fiecare dintre numerele date, construiește lista divizorilor săi mai mari ca 1, apoi, folosind cele n liste de divizori, determină și afișează divizorii comuni ai celor n numere.

Exemplu: pentru $n=4$ și lista inițială $L=(15 \rightarrow 60 \rightarrow 210 \rightarrow 90)$, se vor construi listele $L_1=(3 \rightarrow 5)$, $L_2=(3 \rightarrow 4 \rightarrow 5)$, $L_3=(2 \rightarrow 3 \rightarrow 5 \rightarrow 7)$ și $L_4=(3 \rightarrow 5 \rightarrow 6)$, iar divizorii comuni memorați în cele patru liste de divizori sunt 3 și 5.

28. Fișierul "**fraza.txt**" conține pe un singur rând o frază, alcătuită din cuvinte, între care apar ca separatori spațiul și virgula, fraza încheindu-se cu caracterul "punct". Să se creeze o listă liniară simplu înlănțuită cu cuvintele frazei (câmpul de informație al fiecărui nod va fi de tip șir de caractere și va memora un cuvânt al frazei), apoi, parcurgând lista, să se afișeze cuvântul (cuvintele) de lungime maximă. Nu se vor folosi vectori sau alte structuri alocate static.

Exemplu: Dacă în fișier se găsește fraza "mie îmi plac caii, sunt mort după ei", atunci lista va fi: ("mie" $\rightarrow$ "îmi" $\rightarrow$ "plac" $\rightarrow$ "caii" $\rightarrow$ "sunt" $\rightarrow$ "mort" $\rightarrow$ "după" $\rightarrow$ "ei"), iar cuvintele de lungime maximă ce trebuie afișate sunt: "plac", "caii", "sunt", "mort" și "după".

29. Se citește de la tastatură un șir de caractere alcătuit din cel mult **100** de litere ale alfabetului latin. Șirul se citește caracter cu caracter, citirea încheindu-se prin tastarea caracterului '#', care nu face parte din el. Folosind o listă liniară simplu înlănțuită, în care fiecare nod va memora în câmpul său de informație câte un caracter al șirului, să se afișeze cea mai lungă secvență de litere din șir care, luată individual, constituie un palindrom. Nu se vor utiliza tipul șir de caractere, vectori, sau alte structuri de date alocate static.

Exemplu: Pentru șirul "vbanacmamxmxmamt", secvențele ce constituie palindroame sunt: "ana" și "mamxmam", iar cea mai lungă dintre acestea este "mamxmam".

Indicații. Problema se bazează pe un algoritm de determinare a unui subșir de lungime maximă cu o anumită proprietate dintr-un șir dat. Pentru testarea proprietății de palindrom a unui subșir puteți folosi algoritmul clasic de extragere a cifrelor.

30. Pentru evidența rezultatelor la Bacalaureat obținute de către elevii unei școli se folosește fișierul "**bac.txt**" care conține:

- pe primul rând numărul n de elevi;
- pe fiecare din următoarele n rânduri media și apoi numele unui elev, separate printr-un spațiu.

Folosind o listă liniară simplu înlănțuită, se cere:

- să se afișeze numele elevului (elevilor) care au obținut media cea mai mare;
- să se determine media generală a școlii.

Indicații: Din fișier se citește mai întâi valoarea lui n , apoi într-un ciclu, la fiecare pas: se citește numele și media unui elev de pe un rând al fișierului și se creează un nod al

listei simplu înlănțuite, având drept informații numele și media citite. Pentru determinarea celei mai mari medii, aplicăm algoritmul clasic de maxim în listă, dar pentru că pot fi mai mulți elevi cu media maximă determinată parcurgem încă o dată lista și afișăm acei elevi a căror medie este egală cu maximul. În sfârșit, media generală a școlii este media aritmetică a mediilor elevilor din listă (suma mediilor împărțită la numărul acestora).

31. Un astrolog dorește să efectueze un studiu statistic privitor la influențele astralelor asupra persoanelor care locuiesc în orașul său. Pentru aceasta are nevoie de un eșantion de **n** persoane. El va întocmi o listă care să cuprindă, pentru fiecare dintre aceste persoane, numele, ziua și luna nașterii, simulată cu ajutorul unei liste liniare alocată dinamic. Pornind de la această listă, astrologul nostru trebuie să poată stabili zodiile în care s-au născut persoanele, și apoi să determine câte persoane sunt născute în fiecare zodie. În acest scop, el va construi câte o listă alocată dinamic pentru fiecare zodie în parte, conținând persoanele născute în acea zodie. Realizați un program care să rezolve problema astrologului nostru. Pentru ca astrologul să fie pe deplin mulțumit, vă propunem să-i oferiți un program care să afișeze și listele tuturor persoanelor născute în fiecare zodie, pe lângă numărul acestora. Datele de intrare se citesc din fișierul "**pers.txt**", care conține:

- pe primul rând numărul **n** al persoanelor din eșantion;
- pe fiecare din următoarele **n** rânduri, ziua nașterii, luna nașterii (în cifre) și numele unei persoane, în această ordine, separate prin spații.

32. La ora de educație fizică, profesorul a cerut elevilor unei clase să se alinieze în ce ordine doresc ei. Fiind un mare iubitor al informaticii, el a observat imediat că în șirul de elevi există situații în care un elev de înălțime maximă este așezat lângă unul de înălțime minimă. Scrieți un program care afișează perechile de elevi alăturați în șir care îndeplinesc condiția de mai sus, precum și numărul acestor perechi, apoi rearanjează copiii în șir, astfel încât să se obțină un număr maxim de perechi cu proprietatea respectivă. Șirul de elevi va fi simulat cu ajutorul unei liste în care fiecare nod va memora numele și înălțimea unui elev, și nu se vor folosi nici un fel de structuri de date alocate static (tablouri, înregistrări, etc). Numărul de elevi, precum și numele și înălțimile elevilor, se citesc de la tastatură.

Răspunsuri la testele grilă:

1. c) 2. c) 3. c) 4. c) 5. c) 6. c) 7. c) 8. d) 9. d)

Bibliografie

1. Emanuela Cerchez, Marinel Șerban - Programarea în limbajul C/C++ pentru liceu, Editura Polirom, 2006 (trei volume);
2. Dumitru Fanache – Informatică, manual pentru clasa a XI-a, varianta C++, Editura Gimnasium, 2002;
3. George Daniel Mateescu, Pavel Florin Moraru – Informatica pentru liceu și bacalaureat, materia din clasa a XI-a, Editura Donaris, Sibiu, 2006;
4. Daniela Oprescu, Liana Bejan Ienulescu, Viorica Pătrașcu – Informatică, manual pentru clasa a XI-a, Editura Niculescu, 2002;
5. Dorian Stoilescu – Culegere de C/C++, Editura Radial, Galați, 1998;
6. Dorian Stoilescu - Manual de C/C++ pentru licee, Editura Radial, Galați, 1998;
7. Colectiv autori - Bac 2009: subiecte posibile, Editura Cygnus, 2009;
8. *** - Subiecte bacalaureat 2006, 2007, 2008, 2009.