4. 3 Structuri de date alocate dinamic

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

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

Pentru alocarea dinamică a acestor structuri se utilizează variabilele dinamicce.

Variabile dinamice (pointeri)

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

Declararea unei variabile ponter 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 ponter*.

De exemplu, declararea unei variabile **p** de tip pointer care reţine 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 conţine nici o adresa, adică *ponterul nul*.

Ca operatori specifici privind pointerii amintim:

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

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

Pascal	C/C++

`var x: Integer;`
`  p: ^Integer;`
`Begin`
`  x:=10;`
`  p:=@x;`
`  WriteLn;`
`  Write('p^ este chiar x: ', p^);`
`End.`
	`#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;`
	`}`

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

- funcţia de alocare a spaţiului de memorie pentru o variabilă pointer **p**

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

- funcţia de eliberare a spaţiului de memorie ocupat de o variabilă pointer **p**

Pascal	C	C++
`dispose(p);`	`free(p);`	`delete p;`

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

Pascal	C/C++
`maxavail();`	`void malloc(unsigned nr_octeţi);`

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

Pascal	C/C++

`if sizeof(int)<=maxavail() then`
`  p=new()`
`else`
`  WriteLn(”Zona HEAP plina!”);`
	`if (malloc(sizeof(int))`
	`  p=new int;`
	`else`
	`  cout<<”Zona HEAP plina!”;`

Pentru uşurinţa expunerii, în continuare nu vom face verificări de alocare de spaţiu în heap (dar vom utiliza funcţiile şi operatorii atât pentru C cât şi pentru C++).

4. 3. 1 Liste simplu înlănţuite (definiţie, operaţii specifice descrise în limbaj de programare)

Prin *listă* (sau *listă liniară*) înţelegem un şir finit (eventual vid) de *elemente* (numite şi *noduri* sau *componente*) aparţinând unei mulţimi date $\mathbf{E=\{e_1, e_2, \ldots, e_n\}}$ care au următoarele proprietăţi:

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

Prin urmare, lista este o structură de date între ale carei componente există o *relaţie de ordine* (fiecare element al listei are *un singur succesor*, cu excepţia ultimului element din listă care nu are nici un succesor şi *un singur predecesor*, cu execpţia primului element din listă care nu are nici un predecesor).

Exemple de liste: elementele unui vector, şirul format din numele din cartea de telefon (lista abonaţilor), 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.

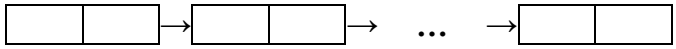
Memorarea listelor se poate face în mai multe moduri, în continuare prezentându-se două dintre ele.

Alocarea *secvenţială* constă în a memora elementele listei în locaţii succesive de memorie, conform ordinei acestor elemente în listă şi în acelaşi 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 *înlănţuită* presupune că fiecare element al listei este înlocuit cu o celulă formată din două părţi: *o parte de informaţie* corespunzătoare elementului şi *o parte de legătură* ce conţine adresa celulei corespunzătoare următorului element (adresă numită obişnuit *pointer*). Ca şi la alocarea secvenţială, mai trebuie memorată o adresă de bază şi, în plus, partea de legătură a ultimei celule primeşte 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

inf	**urm**

cu semnificaţii evidente. Legăturile vor fi precizate cu ajutorul unor săgeţi, 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 informaţia este o dată de tip întreg. În acest caz, implementarea unui nod al listei liniare simple se face astfel:

Pascal

```
type pnod = ^nod;
 nod = record
 info :Integer;
 urm: pnod;
 end;
```

C/C++

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

Operaţiile cele mai importante care se efectuează cu listele sunt:

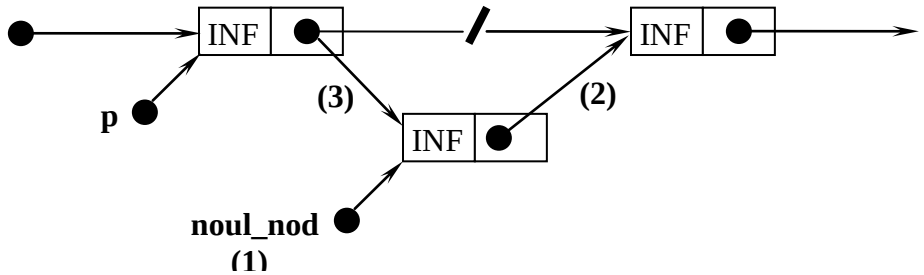
- crearea unei liste;
- parcurgerea şi prelucrarea unei liste;
- adăugarea unui nou element în listă;
- ştergerea unui element din listă;
- întoarcerea numărului de elemente ale listei (cardinalul său).

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

1. lista este vidă şi 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 situaţii:
        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 situaţii:
        i. **p** indică ultimul element din listă;
        ii. **p** nu indică ultimul element din listă.

Pentru a evita aceste situaţii, se poate recurge la un mic artificiu creând lista de la început cu două noduri fictive (în care nu se memorează nici o informaţie) şi 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 adăugam 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 informaţia în câmpul **inf** din **noul_nod**.

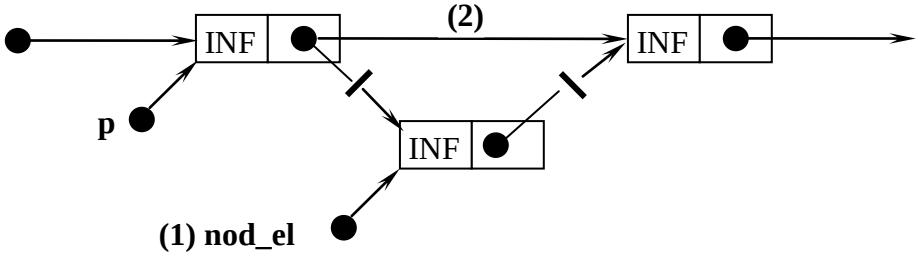
Varianta Pascal		**Varianta C/C++**	
`new(noul_nod);`	`{1}`	`noul_nod=new NOD;`	`// 1`
`noul_nod^.urm:=p^.urm;`	`{2}`	`noul_nod->urm=p->urm;`	`// 2`
`p^.urm:=noul_nod;`	`{3}`	`p->urm=noul_nod;`	`// 3`
`ReadLn(noul_nod^.inf);`	`{4}`	`cin<<noul_nod->inf;`	`// 4`

Facem observaţia că trebuie respectată cu stricteţe ordinea operaţiilor deoarece, de exemplu, dacă s-ar efectua operaţia **3** înaintea operaţirei **2**, s-ar pierde adresa nodului care urma iniţial după **p** (în mod iremediabil). Doar operaţia **4** poate fi efectuată oriunde după operaţia **1** (adică, oricând, după ce s-a alocat spaţiu pentru noul nod).

Într-o listă simplu înlănţuită putem adăuga un nou nod şi *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 informaţia din nodul indicat de **p** (aflat în faţa noului nod) după care noua informaţie o depunem în nodul din faţă, momentan indicat de **p** (după care **p** va memora adresa nodului introdus în listă).

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

Ştergerea 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 şi la adăugarea unui nou nod în listă, şi la *ştergerea* unui nod din listă sunt posibile mai multe situaţii. 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** şi 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 şi 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 înlănţuite* se face astfel:

Pascal

```
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.
```

C/C++

```
#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;
}
```

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

Varianta Pascal

```
p=cap;
while (p<>NIL)do
begin
 Write(p->inf, " "); p=p^.urm
End
```

Varianta C/C++

```
p=cap;
while (p!=NULL)
{
 cout<<p->inf<<" "; p=p->urm;
}
```

adcă iniţializăm pointerul **p** cu adresa capului listei, după care, în mod repetat, afişăm informaţia (sau o prelucrăm, în funcţie de necesităţi) iar prin instrucţiunea **p:=p^urm** în **Pascal**, respectiv **p=p->urm** în **C/C++**, ne poziţionăm pe următoarea înregistrare până când ajungem la ultimul nod unde următoarea adresă este **NIL** în **Pacal**, respectiv

NULL în **C/C++**. Adăugaţi aceste instrucţiuni la sfârşitul programului anterior pentru a afişa lista creată.

Este evident că alocarea înlănţuită necesită mai multă memorie decât cea secvenţială. Trebuie menţionat însă că în aplicaţii partea de informaţie este mult mai mare decât cea de legătură. Pe de altă parte, alocarea înlănţuită elimină necesitatea deplasării de elemente pentru operaţiile de introducere şi de scoatere de elemente din listă. Un alt avantaj al alocării înlănţuite este acela că permite folosirea unui spaţiu de memorare comun pentru mai multe liste, ceea ce implică o economie de memorie, având în vedere numeroasele operaţii de introducere şi de scoatere a elementelor din listă (liste).

În cazul în care se impun anumite restricţii asupra operaţiilor de inserare, ştergere sau consultare a elementelor unei liste liniare, se obţin câteva cazuri perticulare de liste mult folosite în practică: stiva, coada şi lista cu două capete.

Stiva

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

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

Coada

O listă liniară în care inserările elementelor se fac la un capăt al listei (sfârşitul listei), iar consultările şi ştergerile de elemente se fac la celălalt capăt al listei (capul listei) se numeşte *coadă*, *fir de aşteptare* sau *listă FIFO* (First-IN, First-Out, adică primul intrat va fi primul ieşit 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

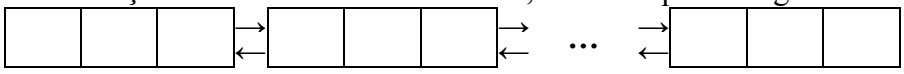
O listă liniară în care inserările, ştergerile şi regăsirea elementelor se fac la ambele capete ale listei.

4. 3. 2 Liste dublu înlănţuite (definiţie, operaţii specifice descrise în limbaj de programare)

Dacă elementele unei liste liniare sunt formate din triplete de forma

ant	**inf**	**urm**

unde **inf** reprezinta informatia nodului, **ant** este adresa către predecesor (nodul anterior), iar **urm** este adresa catre succesor (nodul următor), atunci se obţine o *listă liniară dublu înlănţuită* sau *listă liniară simetrică*, care se reprezintă grafic astfel:

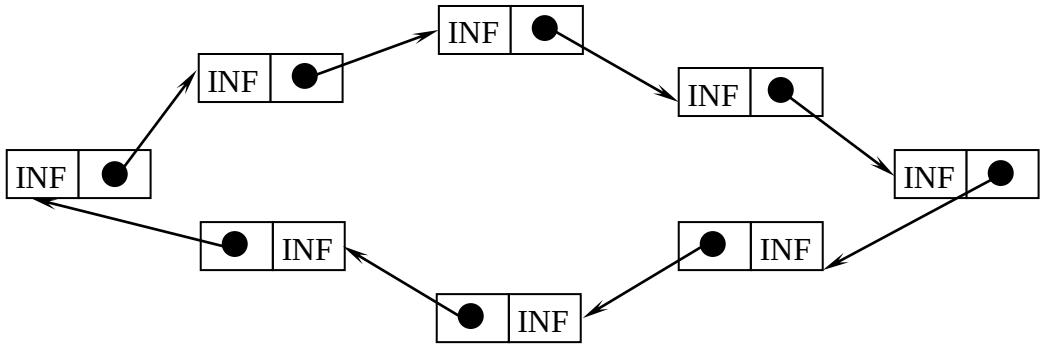


O astfel de listă poate fi parcursă în ambele sensuri, deci prezintă două grade de libertate.

Operaţiile specifice listelor dublu înlănţuite se bazează pe aceeaşi logică aşa cum am văzut la listele simplu înlănţuite, doar că trebuie să ţinem seama că mai există şi legătura spre elementul anterior şi să dăm mare atenţie ordinii în care ralizăm legăturile pentru a nu pierde informaţii.

4. 3. 3 Liste circulare (definiţie, operaţii specifice descrise în limbaj de programare)

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



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

Listele circulare se mai numesc şi *liste închise*, celelalte purtând numelre de *liste deschise*.

La operaţiile specifice listelor circulare trebuie să ţinem cont, în plus, şi de legăturile existente între ultimul nod şi primul nod.

Atragem atenţia asupra greşelilor frecvente în utilizarea alocării dinamice pentru liste:

- nerespectarea succesiunii corecte a operaţiilor;
- prelucrarea (citirea, atribuirea) informaţiilor 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 ştergerii sau inserţiei unui nod nu se actualizează corect legăturile;
- pierderea poziţiei capului listei;
- ştergerea unui nod înaintea preluării (memorării) legăturii următoare;
- inserţia unui nod înaintea găsirii nodului anterior.

4. 3. 4 Teste grilă (de la Bac)

1. Structura de date la care se aplică principiul „primul venit, primul ieşit”: (first in, first out) este:

a. lista înlănţuită **b.** stiva **c.** Coada **d.** Graf orientat

Bacalaureat 2009 (varianta 25, II. 1)

2. Nodurile unei liste dublu înlănţuite reţin în câmpurile **info**, **adp** şi **adu** o informaţie numerică, adresa nodului precedent şi respectiv adresa nodului următor din listă. Ştiind că lista este corect construită şi că două noduri **p** şi **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

Varianta C/C++

a. p->adp==q->adu b. p->adu==q->adu c. p->adp==q d. p->adp==q->adp

Bacalaureat 2006 (Simulare - subiectul I.6)

3. Varianta Pascal

Se consideră o listă simplu înlănţuită ale cărei noduri reţin 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**
Stabiliţi care dintre acestea se utilizează şi în ce ordine:

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

Varianta C/C++

Se consideră o listă simplu înlănţuită ale cărei noduri reţin 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->urm=q;** 2) **q->urm=p;** 3) **p=q->urm;** 4) **q=p->urm;** 5) **p->urm=q->urm;** 6) **q->urm=p->urm; .**
Stabiliţi care dintre acestea se utilizează şi în ce ordine:

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

Bacalaureat 2006 (Simulare - subiectul I.7)

4. Varianta Pascal

Într-o listă simplu înlănţuita, cu cel puţin patru elemente, fiecare element reţine în câmpul **urm** adresa elementului următor din listă. Dacă **p**, **q** şi **r** sunt adresele a trei elemente din listă astfel încât **p^.urm=q^.urm^.urm** şi **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

Varianta C/C++

Lntr-o listă simplu înlănţuită, cu cel puţin patru elemente, fiecare element reţine în câmpul **urm** adresa elementului următor din listă. Dacă **p**, **q** şi **r** sunt adresele a trei elemente din listă astfel încât **p->urm==q->urm->urm** şi **r->urm==q** atunci ordinea logică a elementelor în listă (elementele fund 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. lntr-o listă simplu înlănţuită, cu cel puţin patru elemente, fiecare element reţine 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 şi numai dacă este satisfăcută condiţia:

Varianta Pascal

a. q^.adr^.adr=p b. p^.adr=q
c. p^.adr^.adr=q d. q^.adr=p^.adr^.adr

Varianta C/C++

a. **q->adr->adr==p** **b.** **p->adr==q**
c. **p->adr->adr==q** **d.** **q->adr==p->adr->adr**

Bacalaureat 2007 (Varianta 3, subiectul I.4)

6. Lntr-o listă simplu înlănţuită, cu cel puţin patru elemente, fiecare element reţine în câmpul **urm** adresa elementului următor din listă, iar **p** memorează adresa celui de-al treilea element din listă. Atunci **q** reţine adresa primului element din listă dacă şi numai dacă este satisfăcută condiţia:

Varianta Pascal

a. **p^.urm^.urm=q^.urm** **b.** **p^.urm^.urm=q**
c. **q^.urm^.urm^.urm=p^.urm** **d.** **q^.urm^.urm=p^.urm**

Varianta C/C++

a. **p->urm->urm==q->urm** **b.** **p->urm->urm==q**
c. **q->urm->urm->urm==p->urm** **d.** **q->urm->urm==p->urm**

Bacalaureat 2007 (Varianta 4, subiectul I.5)

7. **Varianta Pascal**

lntr-o listă simplu înlănţuită, cu cel puţin două elemente, fiecare element reţine în câmpul **urm** adresa elementului următor din listă, iar **q** memorează adresa penultimului element din listă. Dacă **p** reţine adresa unui element ce urmează a fi adăugat la sfârşitul listei şi **p^.urm** are valoarea **nil**, stabiliţi care din următoarele este o operaţie corectă de adăugare:

a. **p^.urm=:q** **b.** **q^.urm=:p**
c. **q^.urm^.urm =:p** **d.** **p^.urm^.urm=:q**

Varianta C/C++

lntr-o listă simplu înlănţuită, cu cel puţin două elemente, fiecare element reţine în câmpul **urm** adresa elementului următor din listă, iar **q** memorează adresa penultimului element din listă. Dacă **p** reţine adresa unui element ce urmează a fi adăugat la sfârşitul listei şi **p->urm** are valoarea **NULL**, stabiliţi care din următoarele este o operaţie corectă de adăugare:

a. **p->urm=q** **b.** **q->urm=p**
c. **q->urm->urm=p** **d.** **p->urm->urm=q**

Bacalaureat 2007 (Varianta 5, subiectul I.7)

8. lntr-o listă simplu înlănţuită, cu cel puţin trei elemente, fiecare element reţine în câmpul **nr** un număr întreg şi în câmpul **urm** adresa următorului element din listă. Dacă variabila **prim** reţine adresa primului element din listă, stabiliţi care dintre următoarele secvenţe afişează suma tuturor numerelor memorate în listă mai puţin cele reţinute de primul şi ultimul element:

Varianta Pascal

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

b.
```
s:=0; p:=prim;
while (p^.urm<>nil) do begin s:=s+p^.nr; p:=p.^urm end;
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 înlănţuită alocată dinamic cu cel puţin un element, fiecare element reţine în câmpul **nr** un număr întreg şi în câmpul **urm** adresa următorului element din listă. Ştiind că variabila **p** reţine adresa unui element din listă şi variabila **t** este de acelaşi tip cu variabila **p**, stabiliţi care dintre următoarele secvenţe afişează toate valorile memorate în elementel listei, fiecare valoare fiind afişată exact o dată:

Varianta Pascal

a.
```
t:=p;
while (t^ .urm<>p) do begin
 write(t^.nr, ' '); t:=t^.urm
end
```
b.
```
t:=p;
repeat
 write(t^.nr, ' '); t:=t^.urm
until (t=p)
```
c.
```
t:=p;
while (t<>p) do begin
 write(t^.nr, ' '); t:=t^.urm
end
```
d.
```
t:=p^.urm;
repeat
 write(t^.nr, ' '); t:=t^.urm
until (t=p)
```

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)

4. 3. 5 Probleme rezolvate

1. Într-o listă liniară simplu înlănţuită, alocată dinamic, cu cel puţin **4** elemente, fiecare element reţine 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. Ştiind că variabila **p** reţine adresa primului element din listă, înlocuiţi punctele de suspensie cu expresiile corespunzătoare, astfel încât secvenţa 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)

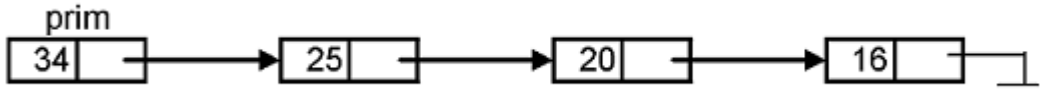
Rezolvare:

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 înlănţuită, alocată dinamic, reţine 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 şi că adresa primului element este reţinută în variabila **prim**, să se scrie declarările de tipuri şi date necesare şi secvenţa de program **PASCAL** (respectiv **C/C++**) care afişează pe ecran numerele memorate în listă, care sunt pătrate perfecte.

Exemplu: pentru lista



se vor afişa numerele `25` şi `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 afişa

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 şi în sens invers şi ştergerea unei liste simplu înlănţuite în mod recursiv (informaţia din noduri este un număr întreg).

Rezolvare. Vom aplica recursivitatea astfel:

■ pentru creare este definită subprogramul recursivă **creare** care întoarce adresa nodului **cap** şi nu are nici un parametru. Mai întâi se va aloca spaţiu pentru nodul carent. Se va introduce informaţia acestuia şi se va interoga utilizatorul dacă doreşte 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** şi nu întoarce nici o valoare. Mai întâi se va tipări informaţia nodului curent şi apoi se vor parcurge celelalte noduri, apelând recursiv subprogramul **parcurgere** având ca parametru nodul următor.

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

■ pentru ştergerea listei se va defini subprogramul **ştergere**. Se va apela mai întâi ştergerea restului listei, apelând recursiv subprogramul **ştergere** şi 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 parcurgere_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 şi produsul a două polinoame folosind liste simplu înlănţuite.

Rezolvare. Pentru aceasta vom face crearea unei liste corespunzătoare coeficienţilor unui polinom. Lista va avea ca informaţie gradul şi coeficientul fiecăruitermen de

coeficient nenul. Vom defini o funcţie **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 acelaşi grad, unul din ele va fi eliminat iar coeficientul celuilalt va lua valoarea sumei coeficienţilor celor doi termeni.

Pentru a calcula suma a două polinoame este suficient să concatenăm listele celor două polinoame într-o a treia listă şi 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 coeficienţilor 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!');
exit
```

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  *creare(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;
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=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=p->urm)
      for(q=p->urm;q;r=q,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=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=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=p->urm)
      for (q=cap2;q;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:=creare;canonic(cap1);
    WriteLn('Polinomul 1:');
    parcurgere(cap1);
    cap2:=creare; 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=creare();canonic(cap1);
   printf("\nPolinomul 1:" );
   parcurgere(cap1);
   cap2=creare(); 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 înlănţuită cu nodurile memorând numere intregi, să se parcurgă atât în sens direct cât şi-n sens invers, dupa care să se şteargă lista. Pentru fiecare operaţie se va scrie câte un subprogram.

Varianta 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;
procedure sterg(p:nod);
begin
 if p<> NIL then
 begin
  sterg(p^.urm);
  dispose(p)
 end
end;
```

Varianta 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("\nParcurgere directa");
 parc(cap1,1);
 puts("\nParcurgere 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('Parcurgere directa');
 parc(cap1,1); WriteLn;
 WriteLn('Parcurgere 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 informaţie utilă a fiecărui nod al stivei acel caracter care se găseşte în poziţia **n** modulo **b**. Apoi, traversând stiva şi afişând resturile, obţinem 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 ghişeu la care se aşteaptă la coadă pentru a fi serviţi mai mulţi clienţi. Să se scrie un program care prelucrează comenzi de forma: **I** (intrarea la coadă), **E** (ieşirea de la coadă în urma servirii de la ghişeu), **L** (listarea persoanelor ce se află la coadă) şi **S** (sfârşitul programului). Clienţii sunt codificaţi 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
```

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=NIL 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=NIL 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=NIL
 end;
 ReadLn; ReadLn
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');
}
```

4. 3. 6 Probleme propuse

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

Indicaţii. Pentru a avea sens inversarea, lista trebuie să aibă cel puţin două noduri. Vom utiliza doi pointeri **p** şi **q** care reţin adresa a căte două noduri succesive şi vom parcurge lista realizând inversarea legăturilor.

2. Să se realizeze următoarele funcţii pentru: crearea unei liste cu cifrele unui număr pe care-l primeşte ca paramtru, adunarea, scăderea, înmulţirea şi împărţirea numerelor memorate astfel în două liste.

3. Să se scrie un subprogram care citeşte pe rând cifrele unui număr întreg foarte mare. Să se utilizeze această funcţie într-un program care citeşte numere foarte mari şi realizează operaţiile de bază cu ele (adunarea, scăderea, înmulţirea, împărţirea).

4. Fiind dată o listă simplu înlănţuită având ca informaţii numere reale, să se afişeze câte numere negative, câte numere egale cu **0** şi câte numere pozitive conţine lista.

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

Indicaţie. Pntru a crea lista ordonată crescător va trebui ca înainte de a insera o nouă valoare în listă să căutăm poziţia sa corectă, apoi să inserăm valoarea pe poziţia corectă, astfel încât la fiecare moment lista să fie sortată. Astfel, dacă lista este vidă sau valoarea care trebuie inserată este mai mică decât informaţia 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 informaţie 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 coeficienţi reali $\mathbf{P(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0}$ poate fi memorat într-o listă simplu înlănţuită în care fiecare nod va reţine coeficientul şi gradul unui monom de forma $\mathbf{a_k x^k}$ al polinomului. Definiţi şi creaţi structurile de date necesare, apoi scrieţi câte un subprogram pentru:

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

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

Indicaţii: Pentru a putea aduna într-o manieră foarte simplă două polinoame este recomandabil ca lista aferentă polinomului **P(x)** să conţină **n** noduri, ce vor memora coeficienţii şi gradele monoamelor. Fie **coef** şi **grad** cele două câmpuri ale fiecărui nod. Dacă un monom lipseşte 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** şi la fiecare pas adăugăm la **v** valoarea unui monom $\mathbf{a_k x^k}$, care se poate exprima sub forma **coef*putere(x, grad)**. Funcţia **putere(b,m)** returnează $\mathbf{b^m}$.

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

7. Fie două *polinoame rare* (în care majoritatea coeficienţilor 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 înlănţuită în care sunt reţinute monoamele în ordinea crescătoare a gradelor, pentru fiecare monom reţinându-se în partea de informaţii coeficientul şi gradul său. Spunem în acest caz că polinoamele sunt memorate în *forma condensată* (adică nu sunt memorate şi monoamele cu coeficienţi 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. Scrieţi un subprogram care, primind ca parametri două liste liniare simplu înlănţuite de numere întregi, afişează reuniunea celor doă liste, fără a folosi tablouri sau

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

9. Scrieţi un subprogram care, primind ca parametri două liste liniare simplu înlănţuite de numere întregi, afişează intersecţia celor doă liste, fără a folosi tablouri sau alte structuri de date alocate static. *Intersecţia* a două liste este alcătuită din cheile lor comune, luate o singură dată.

10. Scrieţi un subprogram care, primind ca parametri două liste liniare simplu înlănţuite de numere întregi, afişează diferenţa celor doă liste, fără a folosi tablouri sau alte structuri de date alocate static. *Diferenţa* a două liste este alcătuită din cheile primei liste care nu se află şi-n a doua listă.

11. Scrieţi un program care să creeze o listă liniară dublu înlănţuită în care nodurile conţin ca informaţie numere întregi şi să realizeze principalele operaţii cu această listă (adăugarea unui nou nod, ştergerea unui nod, căutarea unui nod care conţine ca informaţie o anumită valoare întreagă, etc.).

12. Scrieţi un program care să creeze o listă liniară simplu înlănţuită circulară în care nodurile conţin ca informaţie numere întregi şi să realizeze principalele operaţii cu această listă (adăugarea unui nou nod, ştergerea unui nod, căutarea unui nod care conţine ca informaţie o anumită valoare întreagă, etc.).

13. Scrieţi un program care să creeze o listă liniară dublu înlănţuită circulară în care nodurile conţin ca informaţie numere întregi şi să realizeze principalele operaţii cu această listă (adăugarea unui nou nod, ştergerea unui nod, căutarea unui nod care conţine ca informaţie o anumită valoare întreagă, etc.).

14. Scrieţi un subprogram care, primind ca parametru o listă de numere naturale, afişează 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 afişa numerele 49 şi 169.

15. Construiţi o listă de numere naturale citite din fişierul "**nr.txt**" (în care fiecare nod va conţine drept informaţie un număr din fişier), apoi scrieţi un subprogram care afişează cheile listei a căror sumă a cifrelor este egală cu o valoare dată **s**. Numerele sunt scrise în fişier unul sub altul, fiecare pe câte un rând, şi nu se cunoaşte câte astfel de numere conţine fişierul.

Exemplu: Pentru fişierul ce conţine numerele 5, 123, 14, 85, 1121, 16 şi s=5, se vor afişa numerele 5,14 şi 1121.

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

17. Se citeşte de la tastatură un cuvânt de lungime maxim **80** de caractere. Memorând caracterele cuvântului într-o listă simplu înlănţuită (în care fiecare nod va conţine în câmpul de informaţie un caracter), să se verifice dacă respectivul cuvânt este palindrom. Reamintim: un cuvânt este *palindrom* dacă, citind caracterele sale în ordine invesră, de la dreapta la stânga, obţinem acelaşi cuvânt. De exemplu, cuvântul "cojoc" este palindrom.

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

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

19. Scrieţi un subprogram care verifică dacă două liste liniare simplu înlănţuite 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 acelaşi număr de noduri şi nodurile cu acelaşi număr de ordine în cele două liste conţin aceeaşi informaţie.

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

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

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

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

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

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

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

24. Scrieţi un subprogram care, primind ca parametru un pointer **p** către începutul unei liste liniare simplu înlănţuite 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 citeşte un text caracter cu caracter din fişierul "**fraza.txt**". Textul este scris pe un singur rând în fişier, şi poate conţine orice caractere. Să se construiască o listă liniară dublu înlănţuită care să conţină caracterele distincte din text împreună cu frecvenţele lor de apariţie în cadrul textului, apoi să se afişeze conţinutul listei pe ecran (pe fiecare rând se va scrie câte un caracter distinct şi frecvenţa sa de apariţie, separate printr-un spaţiu.

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

Indicaţii: Introducem separat în listă primul număr. Pentru fiecare din restul numerelor trebuie să căutăm poziţia 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 poziţiei pe care ar ocupa-o noul nod pentru ca lista să rămână sortată crescător după care inserăm nodul cu numărul citit şi corectăm legăturile;
- mai există şi posibilitatea ca numărul pe care-l adăugăm în listă să fie mai mare decât numărul din ultimul nod, situaţie în care adăugăm noul nod la sfârşitul listei.

27. Se citesc de la tastatură **n** numere naturale, care se memorează într-o listă (fiecare nod va conţine drept informaţie unul din cele **n** numere). Scrieţi un program care, pentru fiecare dintre numerele date, construieşte lista divizorilor săi mai mari ca **1**, apoi, folosind cele **n** liste de divizori, determină şi afişează divizorii comuni ai celor **n** numere.

Exemplu: pentru n=4 şi lista iniţială L=(15->60->210->90), se vor construi listele Ll= (3-->5), L2= (3->4->5), L3= (2->3->5->7) şi L4= (3->5->6), iar divizorii comuni memoraţi în cele patru liste de divizori sunt 3 şi 5.

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

Exemplu: Dacă în fişier se găseşte fraza "mie îmi plac caii, sunt mort după ei", atunci lista va fi: ('mie"->"îmi"->"plac"->"caii"->"sunt"->"mort"->"după"->"ei"), iar cuvintele de lungime maximă ce trebuie afişate sunt: "plac", "caii", "sunt", "mort" şi "după".

29. Se citeşte de la tastatură un şir de caractere alcătuit din cel mult **100** de litere ale alfabetului latin. Şirul se citeşte caracter cu caracter, citirea încheindu-se prin tastarea caracterului '**#**', care nu face parte din el. Folosind o listă liniară simplu înlănţuită, în care flecare nod va memora în câmpul său de informaţie câte un caracter al şirului, să se afişeze cea mai lungă secvenţă de litere din şir care, luată individual, constituie un palindrom. Nu se vor utiliza tipul şir de caractere, vectori, sau alte structuri de date alocate static.

Exemplu: Pentru şirul "vbanacmamxmamt", secvenţele ce constituie palindroame sunt: "ana" şi "mamxmam", iar cea mai lungă dintre acestea este "mamxmam".

Indicaţii. Problema se bazează pe un algoritm de determinare a unui subşir de lungime maximă cu o anumită proprietate dintr-un şir dat. Pentru testarea proprietăţii de palindrom a unui subşir puteţi folosi algoritmul clasic de extragere a cifrelor.

30. Pentru evidenţa rezultatelor la Bacalaureat obţinute de către elevii unei şcoli se foloseşte fişierul "**bac.txt**" care conţine:
- pe primul rând numărul **n** de elevi;
- pe fiecare din următoarele **n** rânduri media şi apoi numele unui elev, separate printr-un spaţiu.

Folosind o listă liniară simplu înlănţuită, se cere:

a) să se afişeze numele elevului (elevilor) care au obţinut media cea mai mare;

b) să se determine media generală a şcolii.

Indicaţii: Din fişier se citeşte mai întâi valoarea lui **n.** apoi într-un ciclu, la fiecare pas: se citesc numele şi media unui elev de pe un rând al fişierului şi se creează un nod al

listei simplu înlănţuite, având drept informaţii numele şi media citite. Pentru determinarea celei mai mari medii, aplicăm algoritmul clasic de maxim în listă, dar pentru că pot fi mai mulţi elevi cu media maximă determinată parcurgem încă o dată lista şi afişăm acei elevi a căror medie este egală cu maximul. În sfârşit, media generală a şcolii este media aritmetică a mediilor elevilor din listă (suma mediilor împărţită la numărul acestora).

31. Un astrolog doreşte să efectueze un studiu statistic privitor la influenţele astrelor asupra persoanelor care locuiesc în oraşul său. Pentru aceasta are nevoie de un eşantion de **n** persoane. El va întocmi o listă care să cuprindă, pentru fiecare dintre aceste persoane, numele, ziua şi luna naşterii, 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, şi apoi să determine câte persoane sunt născute în fiecare zodie. In acest scop, el va construi câte o listă alocată dinamic pentru fiecare zodie în parte, conţinând persoanele născute în acea zodie. Realizaţi un program care să rezolve problema astrologului nostru. Pentru ca astrologul să fie pe deplin mulţumit, vă propunem să-i oferiţi un program care să afişeze şi listele tuturor persoanelor născute în fiecare zodie, pe lângă numărul acestora. Datele de intrare se citesc din fişierul "**pers. txt**", care conţine:

- pe primul rând numărul **n** al persoanelor din eşantion;
- pe fiecare din următoarele **n** rânduri, ziua naşterii, luna naşterii (în cifre) şi numele unei persoane, în această ordine, separate prin spaţii.

32. La ora de educaţie 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 şirul de elevi există situaţii în care un elev de înălţime maximă este aşezat lângă unul de înălţime minimă. Scrieţi un program care afişează perechile de elevi alăturaţi în şir care îndeplinesc condiţia de mai sus, precum şi numărul acestor perechi, apoi rearanjează copiii în şir, astfel încât să se obţină un număr maxim de perechi cu proprietatea respectivă. Şirul de elevi va fi simulat cu ajutorul unei liste în care fiecare nod va memora numele şi înălţimea unui elev, şi nu se vor folosi nici un fel de structuri de date alocate static (tablouri, înregistrări, etc). Numărul de elevi, precum şi numele şi înălţimile 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 Şerban - 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 şi bacalaureat, materia din clasa a XI-a, Editura Donaris, Sibiu, 2006;
4. Daniela Oprescu, Liana Bejan Ienulescu, Viorica Pătraşcu – Informatică, manual pentru clasa a XI-a, Editura Niculescu, 2002;
5. Dorian Stoilescu – Culegere de C/C++, Editura Radial, Galaţi, 1998;
6. Dorian Stoilescu - Manual de C/C++ pentru licee, Editura Radial, Galaţi, 1998;
7. Colectiv autori - Bac 2009: subiecte posibile, Editura Cygnus, 2009;
8. *** - Subiecte bacalaureat 2006, 2007, 2008, 2009.