
Dana Lica

Mircea Pașoi

INFORMATICĂ

FUNDAMENTELE PROGRAMĂRII

Culegere de probleme – Pascal și C/C++
pentru clasa a XI-a



Editura L&S SOFT

Copyright 2006 © L&S SOFT

Toate drepturile asupra acestei lucrări aparțin editurii L&S SOFT.

Reproducerea integrală sau parțială a textului din această carte este posibilă doar cu acordul în scris al editurii L&S SOFT.

Editura L&S SOFT:

Adresa: Str. Stânjeneilor nr. 8, bl. 29, sc. A, et. 1, apt. 12, sector 4, București.

Telefon: 021-3321315; 021-6366344; 0722-530390; 0722-573701;

Fax: 021-3321315;

E-mail: tsorin@ls-infomat.ro;

Web Site: www.ls-infomat.ro.

Descrierea CIP a Bibliotecii Naționale a României

LICA, DANA

**Fundamentele programării : culegere de probleme de
informatică - Pascal și C++ pentru clasa a XI-a / Dana Lica, Mircea
Pașoi. - București : Editura L & S Soft, 2006**

ISBN (10) 973-88037-2-1 ; ISBN (13) 978-973-88037-2-5

I. Pașoi, Mircea

004.42(075.35)(076)

Tiparul executat la **S.C. Lumina TIPO s.r.l.**

Str. Luigi Galvani nr. 20 bis, Sector. 4, București, tel./fax: 211.32.60; tel: 212.29.27

E-mail: office@luminatipo.com; www.luminatipo.com

CUPRINS

Capitolul 1

Structuri de date

1.1 Structuri de date alocate dinamic - Liste liniare.....	5
1.1.1 Teste cu alegere multiplă și duală.....	5
1.1.2 Probleme rezolvate.....	19
1.1.3 Probleme propuse.....	31
1.2 Arbori și arborescențe.....	37
1.2.1 Concepte teoretice fundamentale.....	37
1.2.2 Teste cu alegere multiplă și duală.....	43
1.2.3 Probleme rezolvate.....	46
1.2.4 Probleme propuse.....	55
1.3 Structuri de date avansate.....	58
1.3.1 Tabele de dispersie-hash.....	58
1.3.2 Arbori de intervale.....	61
1.3.3 Arbori indexați binar.....	68
1.3.4 Arbori eficienți de căutare-treap-uri.....	71
1.3.5 Probleme propuse.....	74

Capitolul 2

Teoria grafurilor

2.1 Noțiuni introductive.....	79
2.1.1 Terminologie.....	79
2.1.2 Moduri de reprezentare la nivelul memoriei.....	82
2.2 Grafuri orientate și neorientate.....	87
2.2.1 Teste cu alegere multiplă și duală.....	87
2.2.2 Probleme rezolvate.....	91
2.2.3 Probleme propuse.....	124
2.3 Probleme și algoritmi avansați pe grafuri.....	134
2.3.1 Probleme rezolvate.....	134
2.3.2 Probleme propuse.....	160

Capitolul 3

Metode de programare

3.1 Metoda backtracking.....	169
3.1.1 Probleme rezolvate.....	169
3.1.2 Probleme propuse.....	186
3.2 Metoda Divide et Impera.....	191
3.2.1 Probleme rezolvate.....	191
3.2.2 Probleme propuse.....	201
3.3 Metoda programării dinamice.....	204
3.3.1 Probleme rezolvate.....	204
3.3.2 Probleme propuse.....	217
3.4 Metoda Greedy.....	221
3.4.1 Probleme rezolvate.....	221
3.4.2 Probleme propuse.....	232
3.5 Probleme de concurs.....	234
3.5.1 Probleme rezolvate.....	234
3.5.2 Probleme propuse.....	255
<u>Indicații și răspunsuri</u>	265

Structuri de date

1.1. *Structuri de date alocate dinamic - Liste liniare*

1.1.1 Teste cu alegere multiplă și duală

1. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```
type point=^nod;  
      nod=record  
        inf : integer;  
        leg : point; end;
```

```
struct point  
{ int inf;  
  point *leg;  
};
```

Identificați care dintre următoarele funcții returnează un număr întreg ce reprezintă prima valoare strict pozitivă dintr-o listă simplu înlănțuită. Adresa de început a acesteia este trimisă la apel prin intermediul parametrului p . În cazul în care lista nu conține decât valori negative, funcția returnează valoarea 0.

a)
function V(p:point):integer;
begin
 V:=0;
 while (p<>nil) **do begin**
 if (p^.inf>0) **then**
 begin V:=p^.inf; **exit**; **end**;
 p:=p^.leg;
 end;
end;

b)
function V(p:point):integer;
begin
 V:=0;
 while (p^.inf<0) **do** p:=p^.leg;
 V:=p^.inf;
end;

c)
function V(p:point):integer;
begin
 while ((p<>nil)and(p^.inf<=0)) **do**
 p:=p^.leg;

a)
int V(point *p)
{
 for(; p!=NULL; p=p->leg)
 if (p->inf>0)
 return p->inf;
 return 0;
}

b)
int V(point *p)
{
 for(; p->inf<0; p=p->leg);
 return p->inf;
}

c)
int V(point *p)
{
 for (;p && p->inf<=0; p=p->leg);
 if (p)
 return p->inf;
 return 0;
}

```

if (p<>nil) then V:=p^.inf
else V:=0;
end;

```

```

d)
function V(p:point):integer;
var x:integer;
begin
  x:=0;
  while (x=0) do begin
    if (p^.inf>0) then x:=p^.inf;
    p:=p^.leg;
  end;
  V:=x;
end;

```

```

d)
int V(point *p)
{
  int x=0;
  for (;!x; p=p->leg)
    if (p->inf>0)
      x=p->inf;
  return x;
}

```

2. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```

type point=^nod;
      nod=record
        inf : integer;
        leg : point; end;

```

```

struct point
{ int inf;
  point *leg;
};

```

Identificați care dintre următoarele funcții calculează numărul valorilor nule dintr-o listă simplu înlănțuită. Adresa de început a acesteia este trimisă la apel prin intermediul parametrului *p*.

```

a)
function Nr(p:point):integer;
var x:integer;
begin
  x:=0;
  while (p<>nil) do begin
    if (p^.inf=0) then inc(x)
    else p:=p^.leg;
  end;
  Nr :=x;
end;

```

```

b)
function Nr(p:point):integer;
begin
  if p=nil then Nr:=0
  else
    if (p^.inf=0) then
      Nr:=Nr(p^.leg)+1
    else Nr:=Nr(p^.leg)
  end;
end;

```

```

c)
function Nr(p:point):integer;
begin
  if p=nil then Nr:=0

```

```

a)
int Nr(point *p)
{
  int x=0;
  while (p!=NULL)
  { if (!p->inf) x++;
    else p=p->leg;
  }
  return x;
}

```

```

b)
int Nr(point *p)
{
  if (p==NULL) return 0;
  if (!p->inf)
    return Nr(p->leg)+1;
  return Nr(p->leg);
}

```

```

c)
int Nr(point *p)
{
  if (p==NULL) return 0;
  if (!p->inf)
    return Nr(p->leg)+1;
}

```

```

else
  if (p^.inf=0) then
    Nr:=Nr(p^.leg)+1
end;

d)
function Nr(p:point):integer;
var x:integer;
begin
  x:=0;
  repeat
    if (p^.inf=0) then inc(x);
    p:=p^.leg;
  until p=nil;
  Nr :=x;
end;

```

```

d)
int Nr(point *p)
{
  int x=0;
  do {
    x+=!p->inf;
    p=p->leg;
  }while (p!=NULL);
  return x;
}

```

3. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```

type point=^nod;
nod=record
  inf : integer;
  leg : point; end;

```

```

struct point
{ int inf;
  point *leg;
};

```

Funcția *Vm* primește, prin intermediul parametrului *p*, adresa de început a unei liste simplu înlănțuite. Presupunem că la apel *i* se transmite adresa primului nod a unei liste în care sunt reținute în ordine valorile întregi -1, 3, 5, 13, 53, 21, 43, 5. Ce valoare va returna funcția în acest caz?

```

function Vm(p:point):integer;
begin
  if p^.leg=nil then Vm:=p^.inf
  else
    if (p^.inf>Vm(p^.leg)) then
      Vm:=p^.inf
    else Vm:=Vm(p^.leg)
  end;
end;

```

```

int Vm(point *p)
{
  if (p->leg==NULL)
    return p->inf;
  if (p->inf>Vm(p->leg))
    return p->inf;
  return Vm(p->leg);
}

```

- a) -1; b) 5; c) 43; d) 53.

4. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```

type point=^nod;
nod=record
  inf : integer;
  leg : point; end;

```

```

struct point
{ int inf;
  point *leg;
};

```

Funcția *Vm*, declarată în continuare, primește, prin intermediul parametrului *p*, adresa de început a unei liste simplu înlănțuite.

```

function Vm(p:point):boolean;
begin
  if p=nil then Vm:=true
  else
    if (p^.inf>0) and (Vm(p^.leg))
    then Vm:=Vm(p^.leg)
    else Vm:=false;
  end;

```

```

int Vm(point *p)
{
  if (p==NULL)
    return 1;
  if (p->inf>0 && Vm(p->leg))
    return Vm(p->leg);
  return 0;
}

```

Funcția va returna valoarea *True* (pentru varianta Pascal), respectiv 1 (pentru varianta C++) dacă:

- lista este vidă;
- lista are primul element pozitiv;
- lista conține numai valori pozitive;
- lista nu conține valori nule.

5. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```

type point=^nod;
nod=record
  inf : integer;
  leg : point; end;

```

```

struct point
{ int inf;
  point *leg;
};

```

Funcția *Sg* primește, prin intermediul parametrului *p*, adresa de început a unei liste simplu înlănțuite.

```

function Sg(p:point):boolean;
begin
  if p^.leg=nil then Sg:=true
  else
    if (p^.inf * p^.leg^.inf<0)
    then Sg:=Sg(p^.leg)
    else Sg:=false;
  end;

```

```

int Sg(point *p)
{
  if (p->leg==NULL)
    return 1;
  if (p->inf * p->leg->inf<0)
    return Sg(p->leg);
  return 0;
}

```

Funcția *Sg* returnează valoarea *True* (pentru varianta Pascal) respectiv 1, (pentru varianta C++) dacă:

- lista are un singur element;
- lista are ultimul element de semn contrar primului;
- oricare două elemente succesive din listă au același semn;
- oricare două elemente succesive din listă au semne diferite.

6. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```

type point=^nod;
nod=record
  inf : integer;
  leg : point; end;

```

```

struct point
{ int inf;
  point *leg;
};

```


Funcția X primește, prin intermediul parametrilor p și q , adresele de început ale unor liste simplu înlănțuite.

```
function X(p,q:point):boolean;
begin
  if p=nil then
    if q=nil then X:=true
    else X:=false
  else
    X:=X(p^.leg,q^.leg)
end;
```

```
int X(point *p,point *q)
{
  if (p==NULL) {
    if (q==NULL) return 1;
    return 0;
  }
  return X(p->leg,q->leg);
}
```

Funcția X returnează valoarea *False* (pentru varianta Pascal), respectiv 0 (pentru varianta C++) dacă:

- ambele liste sunt vide;
- cele două liste conțin același număr de elemente;
- lista a cărei adresă de început este transmisă prin p conține mai puține elemente;
- lista a cărei adresă de început este transmisă prin q conține mai puține elemente.

7. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```
type point=^nod;
      nod=record
          inf : integer;
          leg : point; end;
```

```
struct point
{ int inf;
  point *leg;
};
```

Funcția V_s primește, prin intermediul parametrului p , adresa de început a unei liste simplu înlănțuite.

```
function Vs(p:point):integer;
begin
  if p=nil then Vs:=0
  else
    if frac(sqrt(p^.inf))=0.0 then
      Vs:=Vs(p^.leg)+p^.inf
    else
      Vs:=Vs(p^.leg)
end;
```

```
int Vs(point *p)
{
  if (p==NULL) return 0;
  if (floor(sqrt(p->inf))==
      sqrt(p->inf))
    return p->inf+Vs(p->leg);
  return Vs(p->leg);
}
```

Ce rezultat va returna funcția V_s , dacă la apel este transmisă adresa de început a unei liste ce memorează valorile 1, 2, 3, 4, 5, 6, 7, 8, 9?

- 19;
- 1;
- 10;
- 14;
- 13.

8. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```
type point=^nod;
      nod=record
          inf : integer;
          leg : point; end;
```

```
struct point
{ int inf;
  point *leg;
};
```

Care dintre următoarele secvențe de instrucțiuni permite crearea unei liste simplu înlănțuite nevidă, ale cărei elemente formează un șir de valori întregi consecutive și a cărei adresă de început este memorată de variabila *Prim*?

a)
new(p); x:=10;
while P<>nil do begin
 p^.inf:=x; x:=x+1; p:=p^.leg
end;
Prim:=p;

b)
read(x); x:=abs(x); Prim:=nil;
while x>0 do begin
 new(p); p^.inf:=x; dec(x);
 p^.leg:=Prim; Prim:=p
end;

c)
x:=30; Prim:=nil;
while x<100 do begin
 p^.inf:=x; x:=x-1;
 p^.leg:=Prim; new(p);
end;

d)
new(p); x:=20; Prim:=nil;
while x<>10 do begin
 p^.inf:=x; x:=x-1; p^.leg:=Prim;
 Prim:=p; new(p);
End;

a)
p=new point; x=10;
for (;p!=NULL;p=p->leg)
 p->inf=x++;
prim=p;

b)
scanf("%d",&x);x=abs(x);
prim=NULL;
while (x>0) {
 p=new point; p->inf=x--;
 p->leg=prim; prim=p;
}

c)
x=30; prim=NULL;
while (x<100) {
 p->inf=x++;
 p->leg=prim; p=new point;
}

d)
p=new point; x=20; prim=NULL;
while (x!=10) {
 p->inf=x--; p->leg=prim;
 prim=p; p=new point;
}

9. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```
type point=^nod;
  nod=record
    inf : integer;
    leg : point; end;
```

```
struct point
{
  int inf;
  point *leg;
};
```

Care dintre următoarele secvențe de instrucțiuni sunt corecte sintactic, știind că variabila *p* aparține tipului *point*?

a)
readln(p);

b)
if p<>nil then writeln(p^.leg);

c)
p:=p^.leg^.leg^.inf;

d)
new(p); p^.leg:=p;

a)
cin>>p;

b)
if (p!=NULL) cin>>p->leg;

c)
p=p->leg->leg->inf;

d)
p=new point; p->leg=p;

10. Considerăm declarația următorului tip de date necesar reprezentării unei liste dublu înlănțuite:

```
type pointd = ^nod;
  nod = record
    inf : integer;
    ls, ld : pointd; end;
```

```
struct pointd
{ int inf;
  pointd *ls, *ld;
};
```

Care dintre următoarele subprograme modifică adresa de început a listei pe care o primește la apel?

a)

```
procedure Print(Prim:pointd);
begin
  while Prim<>nil do begin
    write(Prim^.inf, ' ');
    Prim:=Prim^.ld
  end
end;
```

b)

```
procedure Print(Prim:pointd);
var p:pointd;
begin
  p:=Prim;
  while p^.ld<>nil do begin
    if Prim^.ld<>nil then
      Prim:=Prim^.ld^.ld;
    p:=p^.ld
  end
end;
```

c)

```
procedure Print(var p:pointd);
begin
  while p<>nil do begin
    write(p^.inf, ' ');
    p:=p^.ld;
  end;
end;
```

a)

```
void print(pointd *prim)
{
  for (;prim!=NULL;prim=prim->ld)
    printf("%d ", prim->inf);
}
```

b)

```
void print(pointd *prim)
{
  pointd *p;
  for(p=prim;p->ld;p=p->ld)
    if (prim->ld)
      prim=prim->ld->ld;
}
```

c)

```
void print(pointd *&p)
{
  for (;p!=NULL;p=p->ld)
    printf("%d ", p->inf);
}
```

11. Considerăm declarația următorului tip de date necesar reprezentării unei liste dublu înlănțuite:

```
type pointd = ^nod;
  nod = record
    inf : integer;
    ls, ld : pointd; end;
```

```
struct pointd
{ int inf;
  pointd *ls, *ld;
};
```

Care dintre următoarele subprograme verifică, în mod corect, dacă toate elementele unei liste dublu înlănțuite sunt valori pare?

```

a)
function Ok(p:pointd):boolean;
begin
  if p=nil then Ok:=true
  else
    if odd(p^.inf) then
      Ok:=False
    else Ok:=Ok(p^.ld)
  end;
end;

```

```

b)
function Ok(p:pointd):boolean;
begin
  if p=nil then Ok:=true
  else
    if odd(p^.inf) then
      Ok:= Ok(p^.ls)
    else Ok:= Ok(p^.ld)
  end;
end;

```

```

c)
function Ok(p:pointd):boolean;
begin
  if p=nil then Ok:=true
  else
    if p^.inf and 1=1 then
      Ok:=False
    else Ok:=Ok(p^.ld)
  end;
end;

```

```

d)
function Ok(p:pointd):boolean;
begin
  if odd(p^.inf) then
    Ok:=False
  else Ok:=Ok(p^.ld)
end;
end;

```

```

a)
int ok(pointd *p)
{
  if (p==NULL)
    return 1;
  if (p->inf%2==1)
    return 0;
  return ok(p->ld);
}

```

```

b)
int ok(pointd *p)
{
  if (p==NULL)
    return 1;
  if (p->inf%2==1)
    return ok(p->ls);
  return ok(p->ld);
}

```

```

c)
int ok(pointd *p)
{
  if (p==NULL) return 1;
  if (p->inf&1)
    return 0;
  return ok(p->ld);
}

```

```

d)
int ok(pointd *p)
{
  if (p->inf&1)
    return 0;
  return ok(p->ld);
}

```

12. Considerăm declarația următorului tip de date necesar reprezentării unei liste dublu înălțuite:

```

type pointd=^nod;
nod=record
  inf : integer;
  ls,ld : pointd; end;

```

```

struct pointd
{
  int inf;
  pointd *ls,*ld;
};

```

Funcția *Ok* primește, prin intermediul parametrului *p*, adresa celui de al doilea element al unei liste dublu înălțuite:

```

function Ok(p:pointd):boolean;
begin
  if p^.ld=nil then Ok:=false
  else
    if p^.ls^.inf=p^.ld^.inf then
      Ok:=True
    else Ok:=Ok(p^.ld)
  end;
end;

```

```

int ok(pointd *p)
{
  if (p->ld==NULL)
    return 0;
  if (p->ls->inf==p->ld->inf)
    return 1;
  return ok(p->ld);
}

```

Funcția *Ok* returnează valoarea *True* (pentru varianta Pascal), respectiv 1 (pentru varianta C++) dacă:

- a) oricare două elemente succesive din listă au semne diferite;
- b) lista are toate elementele de valori egale;
- c) există un element în listă ai cărui vecini au valori egale;
- d) lista conține elemente de valori consecutive.

13. Considerăm declarația următorului tip de date necesar reprezentării unei liste dublu înălțuite:

```
type pointd = ^nod;
    nod = record
        inf : integer;
        ls, ld : pointd; end;
```

```
struct pointd
{ int inf;
  pointd *ls, *ld;
};
```

Considerând că subprogramele următoare primesc la apel adresele de început a două liste dublu înălțuite, care dintre ele realizează în mod corect concatenarea celor două liste ?

a)

```
procedure C(p1, p2: pointd);
var p: pointd;
begin
    p := p1;
    while (p^.ld <> nil) do p := p^.ld;
    p^.ld := p2;
    p2^.ls := p;
end;
```

b)

```
procedure C(p1, p2: pointd);
var p: pointd;
begin
    p := p1;
    while (p <> nil) do p := p^.ld;
    p^.ld := p2;
    p2^.ls := p;
end;
```

c)

```
procedure C(p1, p2: pointd);
var p: pointd;
begin
    p := p1;
    if p^.ld <> nil then begin
        repeat
            p := p^.ld;
        until p^.ld = nil;
        p^.ld := p2;
        p2^.ls := p;
    end;
end;
```

a)

```
void C(pointd *p1, pointd *p2)
{
    pointd *p;
    for (p = p1; p->ld; p = p->ld);
    p->ld = p2;
    p2->ls = p;
}
```

b)

```
void C(pointd *p1, pointd *p2)
{
    pointd *p;
    for (p = p1; p; p = p->ld);
    p->ld = p2;
    p2->ls = p;
}
```

c)

```
void C(pointd *p1, pointd *p2)
{
    pointd *p;
    p = p1;
    if (p->ld != NULL)
    {
        do
            p = p->ld;
        while (p->ld != NULL);
        p->ld = p2;
        p2->ls = p;
    }
}
```

```

d)
procedure C(p1,p2: pointd);
var p: pointd;
begin
  p:=p1;
  if p^.ld<>nil then begin
    repeat
      p:=p^.ld;
    until p=nil;
    p^.ld:=p2; p2^.ls:=p;
  end;
end;

```

```

d)
void C(pointd *p1,pointd *p2)
{
  pointd *p;
  p=p1;
  if (p->ld!=NULL)
  {
    do p=p->ld; while (p!=NULL);
    p->ld=p2;
    p2->ls=p;
  }
}

```

14. Considerăm declarația următorului tip de date necesar reprezentării unei liste dublu înălănțuite:

```

type pointd=^nod;
      nod=record
        inf : integer;
        ls,ld : pointd; end;

```

```

struct pointd
{ int inf;
  pointd *ls,*ld;
};

```

Adresa de început a unei liste dublu înălănțuite care memorează numerele naturale consecutive de la 9 la 1 este reținută în variabila *prim*. Ce se va afișa în urma apelului *print(prim^.ld)* - varianta Pascal, respectiv *print(prim->ld)* varianta C++ ?

```

procedure print(p:pointd);
begin
  if p^.ld<>nil then begin
    write(p^.ls^.inf,' ');
    print(p^.ld);
    write(p^.ls^.inf,' ');
  end;
end;

```

```

void print(pointd *p) {
  if (p->ld!=NULL) {
    printf("%d ",p->ls->inf);
    print(p->ld);
    printf("%d ",p->ls->inf);
  }
}

```

- a) 9 8 7 6 5 4 3 3 4 5 6 7 8 9;
- b) 9 8 7 6 5 4 3 2 2 3 4 5 6 7 8 9;
- c) 9 8 7 6 5 4 3 4 5 6 7 8 9;
- d) 9 9.

15. Considerăm declarația următorului tip de date necesar reprezentării unei liste dublu înălănțuite:

```

type pointd=^nod;
      nod=record
        inf : integer;
        ls,ld : pointd; end;

```

```

struct pointd
{ int inf;
  pointd *ls,*ld;
};

```

Fie două liste dublu înălănțuite. Adresa de început a primei liste este reținută de variabila *prim1*, iar adresa ultimului element din lista a doua, de variabila *ultim2*. Știind că cele două liste rețin în ordine valorile 4, 12, 4, 3, 2 respectiv 1, 3, 4, 12, 6 ce valoare va fi afișată în urma apelului *writeln(ver(prim1,ultim2))* / *printf("%dn",ver(prim1,ultim2))*? Definiția funcției *ver* este următoarea:

```

function ver(p,q:pointd):integer;
begin
  if (p<>nil)and(q<>nil) then
    if p^.inf=q^.inf then
      ver:=p^.inf
    else ver:=ver(p^.ld,q^.ls)
    else ver:=0;
end;

```

```

int ver(pointd *p, pointd *q) {
  if (p!=NULL&&q!=NULL) {
    if (p->inf==q->inf)
      return p->inf;
    return ver(p->ld,q->ls);
  }
  return 0;
}

```

a) 4;

b) 3;

c) 12;

d) 0.

16. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```

type point=^nod;
  nod=record
    inf : integer;
    leg : point; end;

```

```

struct point
{ int inf;
  point *leg;
};

```

Identificați care dintre următoarele subprograme verifică, în mod corect, dacă două liste liniare simplu înlănțuite conțin exact aceleași informații. Se consideră că la apel funcțiile primesc, prin intermediul celor doi parametri, adresele de început ale celor două liste:

a)

```

function Ok(p,q:point):boolean;
begin
  if (p<>nil)and(q<>nil)then
    if p^.inf<>q^.inf then
      ok:=false
    else ok:=ok(p^.leg,q^.leg)
    else ok:=true;
end;

```

b)

```

function Ok(p,q:point):boolean;
begin
  if (p=nil)and(q=nil)then
    Ok:=true
  else
    if (p<>nil)and(q<>nil) then
      if p^.inf<>q^.inf then
        ok:=false
      else Ok:=Ok(p^.leg,q^.leg)
      else Ok:=false
    end;
end;

```

c)

```

function Ok(p,q:point):boolean;
begin
  Ok:=true;
  while (p<>nil)and(q<>nil) do

```

a)

```

int ok(point *p,point *q)
{
  if (p!=NULL&&q!=NULL) {
    if (p->inf!=q->inf)
      return 0;
    return ok(p->leg,q->leg);
  }
  return 1;
}

```

b)

```

int ok(point *p,point *q)
{
  if (p==NULL&&q==NULL)
    return 1;
  if (p!=NULL&&q!=NULL)
  {
    if (p->inf!=q->inf)
      return 0;
    return ok(p->leg, q->leg);
  }
  return 0;
}

```

```

begin
  if p^.inf<>q^.inf then
    Ok:=false;
    p:=p^.leg; q:=q^.leg;
  end;
end;

d)
function Ok(p,q:point):boolean;
begin
  Ok:=true;
  while (p<>nil)and(q<>nil) do
  begin
    if p^.inf<>q^.inf then Ok:=false;
    p:=p^.leg; q:=q^.leg;
  end;
  if p<>q then Ok:=false;
end;

```

```

c)
int ok(point *p,point *q)
{
  for(;p&&q;p=p->leg,q=q->leg)
    if(p->inf!=q->inf)
      return 0;
  return 1;
}

```

```

d)
int ok(point *p,point *q)
{
  for(;p&&q;p=p->leg,q=q->leg)
    if(p->inf!=q->inf)
      return 0;
  return p==q;
}

```

17. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```

type point=^nod;
nod=record
  inf : integer;
  leg : point; end;

```

```

struct point
{ int inf;
  point *leg;
};

```

Ce valoare returnează funcția de mai jos în urma apelului $Nr(\text{prim}^{\text{leg}})$, dacă *prim* reprezintă adresa de început a unei liste care memorează, în ordine, valorile 3, 3, 8, 8, -3, 7, 7, 7?

```

Function Nr(p:point):integer;
var x:integer;
begin
  x:=0;
  while(p^.leg<>nil) do begin
    if (p^.inf=p^.leg^.inf) then inc(x);
    p:=p^.leg;
  end;
  Nr:=x;
end;

```

```

int nr(point *p)
{
  int x=0;
  for(;p->leg!=NULL;p=p->leg)
    if (p->inf==p->leg->inf)
      x++;
  return x;
}

```

a) 4;

b) 2;

c) 1;

d) 3.

18. Considerăm declarația următorului tip de date necesar reprezentării unei liste simplu înlănțuite:

```

type point=^nod;
nod=record
  inf : integer;
  leg : point; end;

```

```

struct point
{ int inf;
  point *leg;
};

```


Care dintre următoarele subprograme implementează, în mod corect, operația de căutare a unei valori întregi într-o listă simplu înlănțuită ?

a)

```
function ok(x:integer; p:point)
    :boolean;
var q:point;
begin
    q:=p;
    while q^.inf<>x do q:=q^.leg;
    if q<>nil then ok:=true
    else ok:=false;
end;
```

b)

```
function ok(x:integer; p:point)
    :boolean;
begin
    ok:=false;
    if p^.inf<>x then
        ok:=ok(x,p^.leg)
    else ok:=true;
end;
```

c)

```
function ok(x:integer; p:point)
    :boolean;
begin
    while (p<>nil)and(p^.inf<>x)do
        p:=p^.leg;
    ok:=p<>nil;
end;
```

d)

```
function ok(x:integer; p:point)
    :boolean;
begin
    if p=nil then ok:=false
    else
        if p^.inf<>x then
            ok:=ok(x,p^.leg)
        else ok:=true;
    end;
```

a)

```
int ok(int x, point *p)
{
    point *q;
    for (q=p;q->inf!=x;q=q->leg);
    if (q!=NULL)
        return 1;
    return 0;
}
```

b)

```
int ok(int x, point *p)
{
    if (p->inf!=x)
        return ok(x,p->leg);
    return 1;
}
```

c)

```
int ok(int x, point *p)
{
    while(p!=NULL&& p->inf!=x)
        p=p->leg;
    return p!=NULL;
}
```

d)

```
int ok(int x, point *p)
{
    if (p==NULL) return 0;
    if (p->inf!=x)
        return ok(x,p->leg);
    return 1;
}
```

19. Considerăm declarația următorului tip de date necesar reprezentării unei liste dublu înlănțuite:

```
type pointd=^nod;
    nod=record
        inf : integer;
        ls,ld : pointd;
    end;
```

```
struct pointd
{ int inf;
  pointd *ls,*ld;
};
```

Considerăm p adresa de început a unei liste dublu înălănțuite.

```
1) p^.ld^.ls:=q;
2) q^.inf:=0;
3) p^.ld:=q;
4) p^.ld^.ls^.ld:=p^.ld;
5) new(q);
6) p^.ld^.ls:=p;
```

```
1) p->ld->ls = q;
2) q->inf = 0;
3) p->ld = q;
4) p->ld->ls->ld = p->ld;
5) q = new pointd;
6) p->ld->ls = p;
```

Stabiliți în ce ordine trebuie efectuate instrucțiunile de mai sus astfel încât nodul de la adresa q să reprezinte al doilea element al listei:

a) 5, 2, 1, 4, 3, 6; b) 1, 3, 2, 4, 6, 5; c) 5, 3, 6, 4, 2, 1; d) 5, 1, 4, 2, 3, 6.

20. Considerăm declarația următorului tip de date necesar reprezentării unei liste dublu înălănțuite:

```
type pointd = ^nod;
   nod = record
       inf : integer;
       ls, ld : pointd;
end;
```

```
struct pointd
{ int inf;
  pointd *ls,*ld;
};
```

Considerăm o listă dublu înălănțuită nevidă pentru care variabilele *prim* și *ultim* rețin adresele primului element, respectiv a ultimului element. Considerăm apelul *write(X(prim,ultim))* în Pascal, respectiv *printf("%d", X(prim,ultim))*, unde subprogramul *X* este parțial declarat în continuare:

```
function X(p,u:pointd):integer;
begin
  if (.....) then X := 2
  else
    if (p=u) then X:=1
    else X:=X(p^.ld,u^.ls)+2;
end;
```

```
int X(pointd *p,pointd *u)
{
  if (.....) return 2;
  if (p==u) return 1;
  return X(p->ld,u->ls)+2;
}
```

Identificați care dintre expresiile următoare pot înlocui spațiile punctate astfel încât subprogramul *X* să determine numărul de elemente al listei:

```
a) p^.ld = u^.ls
b) p^.ld = u
c) p = u^.ls
d) p^.ld <> u^.ls
```

```
a) p->ld == u->ls
b) p->ld == u
c) p == u->ls
d) p->ld != u->ls
```

1.1.2 Probleme rezolvate

1. Într-un fișier text se află scrise, pe o singură linie, numere naturale mai mici sau egale cu 200. Realizați o funcție care creează o listă simplu înlănțuită, preluând în ordine valorile din fișier, până la întâlnirea unui număr egal cu precedentul citit. Subprogramul va primi, prin parametrul *s*, un șir de caractere care reprezintă numele fișierului și va returna adresa de început a listei create.

Exemplu: Dacă fișierul text din care se efectuează citirea conține, în ordine, valorile 2 3 1 4 4 6 7 7 8, subprogramul va crea o listă ce conține, în ordine, valorile: 2 3 1 4.

Soluție:

Crearea listei este realizată prin adăugarea succesivă la finalul acesteia a câte unui nou nod (listă tip coadă). Pentru aceasta va fi necesară reținerea succesivă a adresei alocate ultimului nod (*u*). La citirea unei noi valori (*x*) din fișier, se verifică dacă ($u \rightarrow \text{inf} < x$) în Pascal, respectiv ($u \rightarrow \text{inf} \neq x$) în C++. Adresa alocată primului nod (*p*) va fi returnată ca rezultat.

<pre>1 type point=^celula; 2 celula=record 3 inf:integer; leg:point; 4 end; 5 6 function creare(s:string) 7 :point; 8 var u,q,p:point; 9 x:integer; 10 begin 11 assign(f,s); reset(f); 12 new(p); p^.leg:=nil; 13 read(f,p^.inf); 14 u:=p; read(f,x); 15 while (u^.inf<>x) do begin 16 new(q); q^.leg:=nil; 17 q^.inf:=x; u^.leg:=q; 18 u:=q; read(f,x); 19 end; 20 creare:=p; 21 end;</pre>	<pre>struct point { int inf; point *leg; }; point* creare(char *s) { point *p,*q,*u; int x; freopen(s, "r", stdin); p=new point; p->leg=NULL; scanf("%d",&(p->inf)); u=p; scanf("%d", &x); while (u->inf!=x) { q=new point; q->leg=NULL; q->inf=x; u->leg=q; u=q; scanf("%d", &x); } return p; }</pre>
--	--

2. Realizați un program care afișează numerele obținute prin permutări circulare ale cifrelor unui număr natural *x* preluat de la tastatură. Cifrele lui *x* vor fi reținute într-o listă simplu înlănțuită. Vor fi implementate două subprograme :

- subprogramul *Lista*, care permite memorarea cifrelor numărului *x* într-o listă simplu înlănțuită. Subprogramul va returna adresa de început a listei create prin intermediul unui parametru;
- subprogramul *Permut*, care primind la apel, prin intermediul unui parametru, adresa de început a unei liste, realizează transferul primului element la finalul listei, fără a folosi memorie suplimentară. Subprogramul va returna noua adresă de început a listei prin intermediul aceluiași parametru.

Exemplu: Pentru $n = 1234$, se va afișa: 2341, 3412, 4123, 1234.

Soluție:

Vom crea o listă de tip stivă pentru ca ea să conțină cifrele numărului n în ordinea scrierii din baza 10. În această situație, adăugarea unui nou nod se face la începutul listei. Adresa alocată primului nod va fi returnată prin parametrul p .

În cadrul subprogramului *Permut* sunt efectuate următoarele operații:

- parcurgerea listei pentru identificarea adresei ultimului nod (q);
- modificarea adresei reținute de q în câmpul *leg*;
- actualizarea adresei primului nod al listei;
- memorarea constantei *NIL/NULL* în câmpul *leg* al noului ultim nod al listei.

În varianta Pascal, subprogramele *Lista* și *Permut* sunt implementate ca proceduri.

```
1  type point=^celula;
2  celula=record
3    inf:integer; leg:point;
4  end;
5
6  var prim,aux:point;x:longint;
7  procedure Lista(x:longint;
8                  var p:point);
9  var q:point;
10 begin
11   p:=nil;
12   repeat
13     new(q); q^.leg:=p;
14     q^.inf:=x mod 10;
15     x:=x div 10; p:=q;
16   until x=0;
17 end;
18
19 procedure Permut(var p:point);
20 var q:point;
21 begin
22   q:=p;
23   while q^.leg<>nil do
24     q:=q^.leg;
25   q^.leg:=p; p:=p^.leg;
26   q^.leg^.leg:=nil;
27 end;
28
29 procedure afis(p:point);
30 begin
31   while p<>nil do begin
32     write(p^.inf); p:=p^.leg;
33   end; writeln;
34 end;
35
36 begin
37   readln(x);
38   Lista(x,prim);
39   aux:=prim;
40   repeat
41     permut (prim); afis(prim);
42   until aux=prim;
43 end.
```

```
#include <stdio.h>

struct point {
    int inf; point *leg;
} *prim, *aux; long x;

void lista(long x,point *p)
{
    point *q;
    p=NULL;
    do {
        q=new point;
        q->leg=p;
        q->inf=x%10;
        x/=10; p=q;
    } while (x);
}

void permut(point *p)
{
    point *q;
    q=p;
    for (;q->leg!=NULL;q=q->leg);
    q->leg=p;p=p->leg;
    q->leg->leg=NULL;
}

void afis(point *p)
{
    for (;p!=NULL;p=p->leg)
        printf("%d",p->inf);
    printf("\n");
}

void main()
{
    scanf("%ld",&x);
    lista(x,prim);
    aux=prim;
    do {
        permut(prim); afis(prim);
    } while (aux!=prim);
}
```

3. Într-o listă simplu înlănțuită care memorează întregi ordonați crescător se dorește inserarea unei noi valori astfel încât monotonia valorilor din listă să se păstreze. Realizați un subprogram care efectuează această operație. Atât adresa de început a listei, cât și valoarea de inserat vor fi primite de subprogram la apel, prin intermediul a doi parametri.

Soluție:

Operația de inserare a unui nou nod în listă presupune efectuarea a două operații:

- identificarea-căutarea nodului înaintea căruia se va face inserarea în listă;
- alocarea unei adrese pentru noul nod și refacerea legăturilor pentru inserarea acestuia în listă. Dacă noul nod se va insera în fața primului, atunci adresa de început a listei se modifică.

În varianta Pascal, subprogramul *inse* este implementat ca procedură.

<pre> 1 type point=^celula; 2 celula=record 3 inf:integer; leg:point; 4 end; 5 6 procedure inse(var p:point; 7 x:integer); 8 var u,q,t:point; 9 begin 10 q:=p; 11 while (q^.inf<x)and(q<>nil)do 12 begin 13 u:=q; 14 q:=q^.leg; 15 end; 16 new(t); 17 t^.inf:=x; 18 if q=p then begin 19 t^.leg:=p; 20 p:=t 21 end 22 else begin 23 u^.leg:=t; 24 t^.leg:=q; 25 end; 26 end;</pre>	<pre> struct point { int inf; point *leg; }; void inse(point *&p, int x) { point *u, *q, *t; q=p; while(q->inf<x&&q!=NULL) { u=q; q=q->leg; } t = new point; t->inf=x; if (q==p) { t->leg=p; p=t; } else { u->leg=t; t->leg=q; } }</pre>
--	---

4. Considerăm o listă simplu înlănțuită în care sunt reținute valori naturale distincte. Să se realizeze un subprogram care șterge elementul din listă ce conține cel mai mare număr prim. Subprogramul va primi la apel, prin intermediul unui parametru, adresa de început a listei.

Exemplu: Considerăm lista ce memorează valorile 3, 1, 2, 8, 14, 9, 7, 5. Subprogramul va elimina din listă elementul ce memorează valoarea 7.

Soluție:

Operația de ștergere a unui nou nod din listă presupune efectuarea a două operații:

- identificarea-căutarea nodului ce va fi șters;
- refacerea legăturilor în listă și eliberarea memoriei. Dacă nodul șters este chiar primul, atunci adresa de început a listei se modifică.

Considerăm că există definită funcția *Ok* care verifică dacă valoarea naturală primită ca parametru este un număr prim. Ea returnează în Pascal valoarea *True* sau *False*, respectiv 0 sau 1 în C++.

În varianta Pascal, subprogramul *sterg* este implementat ca procedură.

```
1  type point=^celula;
2  celula=record
3    inf:integer; leg:point;
4  end;
5  .....
6  procedure sterg(var p:point);
7  var q,t:point; max:longint;
8  begin
9    q:=p; t:=nil;
10   if not ok(p^.inf) then max:=0
11   else max:= p^.inf;
12   while (q^.leg<>nil) do begin
13     if ok(q^.leg^.inf) and
14     (max<q^.leg^.inf) then begin
15       t:=q;
16       max:=q^.leg^.inf;
17     end;
18     q:=q^.leg;
19   end;
20   if max<>0 then
21     if t=nil then begin
22       t:=p;
23       p:=p^.leg;
24       dispose(t);
25     end
26     else begin
27       q:=t^.leg;
28       t^.leg:=t^.leg^.leg;
29       dispose(q);
30     end;
31   end;
```

```
struct point {
    int inf;
    point *leg;
};
.....
void sterg(point *&p)
{
    point *q, *t; long max;
    q=p; t=NULL;
    if (!ok(p->inf)) max=0;
    else max=p->inf;
    for(; q->leg!=NULL; q=q->leg)
        if (ok(q->leg->inf) &&
            max<q->leg->inf)
        {
            t=q;
            max=q->leg->inf;
        }
    if (max!=0) {
        if (t==NULL) {
            t=p;
            p=p->leg;
            delete t;
        } else {
            q=t->leg;
            t->leg=t->leg->leg;
            delete q;
        }
    }
}
```

5. Considerăm o listă simplu înlănțuită care reține cifrele unui număr natural. Să se realizeze o funcție care primește la apel, printr-un parametru, adresa de început a unei astfel de liste și returnează cel mai mare număr care se poate forma cu cifrele distincte, memorate în listă.

Exemplu: Dacă lista primită conține cifrele 3, 1, 2, 2, 2, 3, 8, 8, 1, 9, 9, 9, atunci subprogramul va returna numărul 98321.

Soluție:

Funcția realizează operația cerută printr-o parcurgere a listei și o marcarea succesivă a cifrelor memorate la fiecare adresă. La finalul parcurgerii, în vectorul *ap* elementul de indice *i* are valoarea *True* / 1 (Pascal / C++), dacă cifra *i* se găsește în listă.

<pre>1 type point=^celula; 2 celula=record 3 inf:integer; leg:point; 4 end; 5 6 function nr(p:point):longint; 7 var ap:array[0..9]of boolean; 8 x,i:longint; 9 begin 10 fillchar(ap,sizeof(ap),false); 11 while p<>nil do begin 12 ap[p^.inf]:=true; 13 p:=p^.leg; 14 end; 15 x:=0; 16 for i:=9 downto 0 do 17 if ap[i] then x:=x*10+i; 18 nr:=x; 19 end;</pre>	<pre>struct point { int inf; point *leg; }; long nr(point *p) { char ap[10]; long x,i; memset(ap,0,sizeof(ap)); for(;p!=NULL;p=p->leg) ap[p->inf]=1; x=0; for (i=9;i>=0;i--) if (ap[i]) x=x*10+i; return x; }</pre>
--	--

6. Considerăm o listă simplu înlănțuită ce memorează valori întregi. Realizați un subprogram care, primind la apel, prin intermediul unui parametru, adresa de început a unei astfel de liste, separă elementele în două liste, una formată din valorile pare, iar cealaltă, din valorile impare. Subprogramul va returna adresele de început ale celor două liste create prin intermediul a doi parametri.

Exemplu: Dacă lista primită conține valorile 3, 1, 2, 2, 4, 7, 8, 1 atunci subprogramul va separa valorile în două liste ce conțin numerele 3, 1, 7, 1, respectiv 2, 2, 4, 8.

Soluție:

Separarea nodurilor în două liste se face simultan cu parcurgerea listei inițiale. Variabilele *s1* și *s2* reprezintă *santinele* ale listelor nou obținute.

Santinela este un nod fără informație a cărui adresă reprezintă adresa temporară de început a unei liste. Practic, la finalul prelucrării, câmpul *leg* al santineleni memorează adresa reală de început a listei respective.

Pentru a putea realiza separarea nodurilor, se vor reține, în variabilele *u1* și *u2*, adresele ultimelor elemente ale celor două liste. La fiecare pas, un nod al listei inițiale este adăugat fie la finalul listei ce începe de la adresa *s1*, fie la cea care începe de la adresa *s2*.

În varianta Pascal, subprogramul *split* este implementat ca procedură.

```

1  type point:=^celula;
2  celula=record
3    inf:integer; leg:point;
4  end;
5  .....
6  procedure split(p:point;
7    var p1,p2:point);
8  var s1,s2,u1,u2:point;
9  begin
10   new(s1); new(s2);
11   u1:=s1; u2:=s2;
12   while p<>nil do
13     if odd(p^.inf) then begin
14       u1^.leg:=p; u1:=p;
15       p:=p^.leg; u1^.leg:=nil;
16     end
17     else begin
18       u2^.leg:=p; u2:=p;
19       p:=p^.leg; u2^.leg:=nil;
20     end;
21   p1:=s1^.leg; p2:=s2^.leg;
22   dispose(s1); dispose(s2);
23 end;

```

```

struct point {
  int inf;
  point *leg;
};
.....
void split(point *p,
  point *&p1, point *&p2)
(point *s1,*s2,*u1,*u2;
s1=new point;
s2=new point;
u1=s1; u2=s2;
while (p!=NULL)
  if(p->inf%2==1) {
    u1->leg=p; u1=p;
    p=p->leg; u1->leg=NULL;
  } else {
    u2->leg=p; u2=p;
    p=p->leg; u2->leg=NULL;
  }
p1=s1->leg; p2=s2->leg;
delete s1;
delete s2;
}

```

7. Considerăm o listă simplu înlănțuită ce memorează valori întregi. Realizați un subprogram care, primește la apel, prin intermediul unui parametru p , adresa de început a unei astfel de liste și efectuează ștergerea nodurilor de informații negative. Subprogramul va întoarce noua adresă de început a listei tot prin parametrul p .

Exemplu: Dacă lista conține valorile -3, -1, 2, 6, -4, -7, 8, atunci funcția va returna adresa de început a listei ce conține valorile 2, 6, 8.

Soluție:

Ștergerea succesivă a nodurilor se va face simultan cu operația de parcurgere a listei. Pentru că este posibil ca primul nod al listei să fie șters în mod repetat, s-a folosit o santinelă. Aceasta a fost adăugată la începutul listei adică, înaintea nodului de la adresa transmisă la apel prin parametrul p .

În timpul parcurgerii se păstrează, în variabila u , adresa nodului situat în listă înaintea nodului curent (memorat adresa p). Aceasta este necesară pentru refacerea legăturilor înaintea ștergerii propriu-zise (eliberării adresei de memorie respective).

În varianta Pascal, subprogramul *del* este implementat ca procedură.

```

1  type point:=^celula;
2  celula=record
3    inf:integer; leg:point;
4  end;
5  .....
6  procedure del(var p:point);
7  var s,u,t:point;
8  begin
9    new(s); s^.leg:=p; u:=s;

```

```

struct point {
  int inf;
  point *leg;
};
.....
void del(point *&p)
{
  point *s,*u,*t;
  s=new point; s->leg=p; u=s;

```



```

10 while p<>nil do
11   if p^.inf<0 then begin
12     t:=p; u^.leg:=p^.leg;
13     p:=p^.leg; dispose(t);
14   end
15   else begin
16     u:=u^.leg; p:=p^.leg end;
17   p:=s^.leg; dispose(s);
18 end;

```

```

while (p!=NULL)
if(p->inf<0)
{t=p; u->leg=p->leg;
p=p->leg; delete t;
} else {
u=u->leg; p=p->leg;
}
p=s->leg; delete s;
}

```

8. Considerăm o listă simplu înlănțuită ce memorează valori întregi. Realizați un subprogram care primește la apel, prin intermediul unui parametru, adresa de început a unei astfel de liste și efectuează inserarea cheii minime din listă după fiecare element ce conține o valoare pozitivă.

Exemplu: Dacă lista conține valorile -3, -1, 2, 6, -4, -7, 8 atunci, după executarea subprogramului, aceasta va conține valorile -3, -1, 2, -7, 6, -7, -4, -7, 8, -7.

Soluție:

Funcția *Min* determină valoarea minimă memorată într-o listă a cărei adresă de început o primește prin parametrul *p*.

Inserarea succesivă a unor noi noduri se face simultan cu operația de parcurgere a listei.

În parcurgere se verifică dacă elementul curent memorează o valoare pozitivă, caz în care se va insera un nou nod, altfel se trece la nodul următor. După inserarea unui nod, adresa curentă devine adresa nodului situat după cel inserat.

În varianta Pascal, subprogramul *ins* este implementat ca procedură.

```

1 type point=^celula;
2 celula=record
3   inf:integer; leg:point;
4 end;
5 .....
6 function min(p:point):integer;
7 var m:integer;
8 begin
9   m:=p^.inf;
10  while p<>nil do begin
11    if p^.inf<m then m:=p^.inf;
12    p:=p^.leg;
13  end;
14  min:=m;
15 end;
16 procedure ins(p:point);
17 var q:point; x:integer;
18 begin
19   x:=min(p);
20   while p<>nil do begin
21     if p^.inf>0 then begin
22       new(q); q^.inf:=x;
23       q^.leg:=p^.leg;
24       p^.leg:=q; p:=p^.leg^.leg;
25     end
26     else p:=p^.leg
27   end; end;

```

```

struct point {
  int inf;
  point *leg;
};
.....
int min(point *p)
{int m=p->inf;
for (;p!=NULL;p=p->leg)
  if (m>p->inf)
    m=p->inf;
return m;
}

void ins(point *p)
{point *q;
int x;
x=min(p);
while (p!=NULL)
  if (p->inf>0)
  {q=new point;
q->inf=x;
q->leg=p->leg;
p->leg=q;
p=p->leg->leg;
}
else p=p->leg;
}
}

```

9. Se consideră două liste simplu înlănțuite ce conțin valori întregi ordonate crescător. Realizați un subprogram care interclasează cele două liste, fără a folosi memorie suplimentară. Subprogramul va primi la apel, adresele de început a celor două liste prin intermediul a doi parametri și va returna adresa de început a listei obținute prin intermediul celui de al treilea parametru.

Exemplu: Dacă cele două liste conțin valorile 2, 4, 6, 8, 10, respectiv 3, 5, 11, 13, subprogramul va returna adresa de început a listei ce conține elementele 2, 3, 4, 5, 6, 8, 10, 11, 13.

Soluție:

Parametrii *p1* și *p2* primesc la apel adresele de început a celor două liste. Subprogramul va returna prin parametrul *p*, adresa de început a listei nou obținute. Variabila locală *u* va reține adresa ultimului nod adăugat la lista nou obținută. Cele două liste vor fi parcurse simultan, nodul curent de informație minimă fiind transferat la finalul listei noi.

În varianta Pascal, subprogramul *incl* este implementat ca procedură.

```

1  type point=^celula;
2  celula=record
3    inf:integer; leg:point;
4  end;
5  .....
6  procedure incl(p1,p2:point;
7                var p:point);
8  var u:point;
9  begin
10   if (p1^.inf<p2^.inf) then
11   begin
12     p:=p1; u:=p1; p1:=p1^.leg
13   end
14   else begin
15     p:=p2; u:=p2;
16     p2:=p2^.leg
17   end;
18   while (p1<>nil)and(p2<>nil)
19   do
20     if p1^.inf<p2^.inf then
21     begin
22       u^.leg:=p1;
23       u:=p1;
24       p1:=p1^.leg;
25     end
26     else begin
27       u^.leg:=p2;
28       u:=p2;
29       p2:=p2^.leg;
30     end;
31     if p1<>nil then u^.leg:=p1
32     else u^.leg:=p2;
33   end;

```

```

struct point {
  int inf;
  point *leg;
};
.....
void incl(point *p1,
          point *p2,point *&p)
{
  point *u;
  if (p1->inf<p2->inf)
  {p=p1; u=p1;
   p1=p1->leg;
  } else
  {p=p2;
   u=p2;
   p2=p2->leg;
  }
  while (p1!=NULL&& p2!=NULL)
  if (p1->inf < p2->inf)
  {
    u->leg=p1;
    u=p1;
    p1=p1->leg;
  } else {
    u->leg=p2;
    u=p2;
    p2=p2->leg;
  }
  if (p1!=NULL) u->leg=p1;
  else u->leg=p2;
}

```

10. Realizați un subprogram care implementează operația de adunare pe numere mari ($a=b$). Cifrele fiecărui număr vor fi reținute în ordine inversă în câte o listă simplu înlănțuită. Subprogramul va primi la apel, prin intermediul parametrilor p și r , adresele de început ale celor două liste. Tot prin parametrul p se va returna adresa listei ce reprezintă suma.

Exemplu: Pentru numerele $a=188$ și $b=9929$, cele două liste rețin, în ordine, cifrele 8, 8, 1, respectiv 9, 2, 9, 9. Subprogramul va returna, prin parametrul p , adresa de început a listei ce conține 7, 1, 1, 0, 1, adică reprezentarea numărului 10117.

Soluție:

Subprogramul implementează algoritmul cunoscut de adunare a numerelor mari. Listele sunt parcurse simultan, cifra reținută în prima listă devenind restul modulo 10 a sumei dintre cifrele corespunzătoare din cele două numere și restul anterior(t).

Dacă o listă s-a epuizat, atunci algoritmul procesează lista care conține încă cifre.

În varianta Pascal, subprogramul *adun* este implementat ca procedură.

```

1  type point=^celula;
2  celula=record
3    inf:integer; leg:point;
4  end;
5  .....
6  procedure adun(var p:point;
7                 r:point);
8  var q,q1,u:point;t:integer;
9  begin
10   q:=p; t:=0;
11   while (q<>nil)and(r<>nil)do
12   begin
13     q^.inf:=q^.inf+r^.inf+t;
14     t:=q^.inf div 10;
15     q^.inf:=q^.inf mod 10;
16     u:=q;
17     q:=q^.leg;
18     r:=r^.leg;
19   end;
20   if r<>nil then begin
21     u^.leg:=r; q:=r; end;
22   while (q<>nil) do begin
23     q^.inf:=q^.inf + t;
24     t:=q^.inf div 10;
25     q^.inf:=q^.inf mod 10;
26     u:=q; q:=q^.leg;
27   end;
28   if t<>0 then begin
29     new(q1); q1^.inf:=t;
30     q1^.leg:=nil; u^.leg:=q1;
31   end;
32 end;
```

```

struct point {
  int inf;
  point *leg;
};
.....
void adun(point *&p,point *r)
{
  point *q,*q1,*u; int t;
  q=p; t=0;
  while (q!=NULL&&r!=NULL)
  {
    q->inf+=r->inf+t;
    t=q->inf/10;
    q->inf%=10;
    u=q;
    q=q->leg;
    r=r->leg;
  }
  if (r!=NULL) {
    u->leg=r; q=r;
  }
  while (q!=NULL) {
    q->inf+=t;
    t=q->inf/10;
    q->inf%=10;
    u=q; q=q->leg;
  }
  if (t!=0) {
    q1=new point; q1->inf=t;
    q1->leg=NULL; u->leg=q1;
  }
}
```

11. Considerăm o listă circulară în care sunt memorate valori întregi. Să se realizeze o funcție care elimină, începând cu un anumit element, din k în k elemente. Operația de ștergere se oprește când lista conține un singur element. Valoarea lui k și adresa elementului de la care se începe ștergerea sunt transmise la apel prin doi parametri. Funcția va returna adresa ultimului element rămas în listă.

Exemplu: Dacă, începând cu adresa de unde începe ștergerea, lista conține valorile 3, 1, 2, 6, 4, 7, 8 și $k=3$, atunci funcția va întoarce adresa la care este memorată valoarea 6. S-au eliminat în ordine valorile 2, 7, 1, 8, 4 și 3.

Soluție:

Algoritmul parcurge lista și efectuează ștergerea până când adresa curentă p memorează în câmpul *leg* adresa p , deci lista circulară conține un singur element. Ștergerea este realizată și cu eliberarea memoriei.

```

1  type point=^celula;
2  celula=record
3    inf:integer; leg:point;
4  end;
5  .....
6  function del(var p:point;
7                k:byte):point;
8  var i:integer;t:point;
9  begin
10   while p^.leg<>p do begin
11     for i:=1 to k-2 do
12       p:=p^.leg;
13     t:=p^.leg;
14     p^.leg:=p^.leg^.leg;
15     p:=p^.leg;
16     dispose(t);
17   end;
18   del:=p;
19 end;
```

```

struct point {
  int inf;
  point *leg;
};
.....
point* del(point *&p,
            unsigned char k)
{
  int i; point *t;
  while (p->leg!=p)
  {
    for (i=0;i<2<k;i++)p=p->leg;
    t=p->leg;
    p->leg=p->leg->leg;
    p=p->leg;
    delete t;
  }
  return p;
}
```

12. Realizați un subprogram *Perm* care afișează toate permutările circulare ale unui șir de valori întregi reținute într-o listă circulară. Subprogramul primește, la apel, adresa unui element din listă, considerat ca element de început. El apelează subprogramul *Print* care afișează elementele unei liste circulare, începând de la o anumită adresă primită ca parametru.

Exemplu: Pentru lista circulară care conține valorile 1, 2, 3 subprogramul *Perm* va afișa: 1, 2, 3; 2, 3, 1; 3, 1, 2

Soluție:

În cadrul subprogramului *perm* se efectuează parcurgerea listei circulare. Adresa fiecărui nod al listei va constitui parametru actual (efectiv) la apelul subprogramului *print*.

Ambele subprograme sunt implementate în Pascal ca proceduri.

```

1  type point=^celula;
2  celula=record
3    inf:integer; leg:point;
4  end;
5  .....
6  procedure print(p:point);
7  var t:point;
8  begin
9    t:=p;
10   repeat
11     write(p^.inf, ' ');
12     p:=p^.leg;
13   until p=t;
14   writeln;
15 end;
16
17 procedure perm(p:point);
18 var t:point;
19 begin
20   t:=p;
21   repeat
22     print(p);
23     p:=p^.leg;
24   until p=t;
25 end;

```

```

struct point {
  int inf;
  point *leg;
};
.....
void print(point *p)
{
  point *t;
  t=p;
  do {
    printf("%d ", p->inf);
    p=p->leg;
  } while (p!=t);
  printf("\n");
}

void perm(point *p)
{
  point *t;
  t=p;
  do {
    print(p);
    p=p->leg;
  } while (p!=t);
}

```

13. Se consideră un șir de valori reale reținute într-o listă dublu înălțuită. Realizați un subprogram care permite ștergerea din listă a elementelor cu număr de ordine par (al doilea, al patrulea, ș.a.m.d). Subprogramul va primi, la apel, adresa de început a listei.

Exemplu: Lista care reține valorile 1.2, 2.0, 3.2, 4.56, 7.78 va avea, în urma executării subprogramului, valorile 1.2, 3.2, 7.78.

Soluție:

Ștergerea succesivă a nodurilor se va face simultan cu operația de parcurgere a listei.

Deoarece lista este dublu înălțuită, la operația de ștergere este necesară refacerea legăturilor memorate în ambii vecini ai nodului curent. Ștergerea este realizată cu eliberarea adresei de memorie respective.

În varianta Pascal, subprogramul *del* este implementat ca procedură.

```

1  type pointd=^celula;
2  celula=record
3    inf:double;
4    ls,ld:pointd;
5  end;
6  .....
7  procedure del(p:pointd);
8  var t,q:pointd;
9  begin
10   q:=p^.ld;

```

```

struct pointd
{ double inf;
  pointd *ls, *ld;
};
.....
void del(pointd *p)
{
  pointd *t,*q;
  q=p->ld;

```

```

11 while q<>nil do begin
12   t:=q;
13   q^.ls^.ld :=q^.ld;
14   q^.ld^.ls :=q^.ls;
15   if q^.ld<>nil then
16     q:=q^.ld^.ld
17   else q:=nil;
18   dispose(t);
19 end;
20 end;

```

```

while(q!=NULL)
{t=q;
q->ls->ld=q->ld;
q->ld->ls=q->ls;
if (q->ld!=NULL)
q=q->ld->ld;
else q=NULL;
delete t;
}
}

```

14. Realizați un subprogram care inversează legăturile într-o listă simplu înlănțuită fără a utiliza memorie suplimentară. În acest fel primul element al listei va deveni ultimul. Subprogramul va primi ca parametru la apel, adresa primului element.

Exemplu: Lista care reține valorile 1, 2, 3, 4 va avea, în urma executării subprogramului, valorile 4, 3, 2, 1.

Soluție:

Implementarea recursivă permite memorarea în stivă a nodului următor nodului curent, transmis prin parametrul p . Reținerea acestei adrese se face prin intermediul variabilei locale q . La revenirea din recursivitate se realizează inversarea legăturilor.

```

1 type point=^celula;
2 celula=record
3   inf:integer; leg:point;
4 end;
5 .....
6 function inv(p:point):point;
7 var q:point;
8 begin
9   if p^.leg=nil then inv:=p
10  else begin
11    q:=p^.leg;
12    inv:=inv(p^.leg);
13    q^.leg:=p;
14    p^.leg:=nil;
15  end;
16 end;

```

```

struct point {
  int inf;
  point *leg;
};
.....
point* inv(point *p)
{
  point *q,*t;
  if (p->leg=NULL)
    return p;
  q=p->leg;
  t=inv(p->leg);
  q->leg=p;
  p->leg=NULL;
  return t;
}

```

15. Considerăm un șir de n numere naturale. Pentru fiecare valoare citită se dorește reținerea într-o listă simplu înlănțuită a tuturor factorilor primi care apar în descompunerea ei. Realizați un subprogram care permite citirea celor n numere și care memorează, într-un tablou unidimensional, fiecare adresă de început a listelor create.

Soluție:

Structura de date folosită permite indexarea adreselor de început a listelor create. Practic, în vectorul a , elementul $a[k]$ reprezintă adresa de început a listei generate de al k -lea număr citit. Dacă această listă este parcursă, atunci pot fi afișați

toți factorii primi care apar în descompunerea celui de al k -lea număr citit. Aceste liste sunt create sub forma unor stive, deoarece fiecare nou factor este plasat, în lista corespunzătoare, înaintea primului nod.

<pre> 1 type point=^celula; 2 celula=record 3 inf:integer; 4 leg:point; 5 end; 6 sir= array[1..100]of point; 7 8 procedure Vec_L(var a:sir); 9 var x,i,j,e:integer; 10 p:point; 11 begin 12 readln(n); 13 for i:=1 to n do begin 14 a[i]:=nil; read(x); j:=2; 15 while x<>1 do begin 16 e:=0; 17 while x mod j=0 do begin 18 inc(e); x:=x div j; 19 end; 20 if e>0 then begin 21 new(p); p^.inf:=j; 22 p^.leg:=a[i]; a[i]:=p; 23 end; 24 inc(j); 25 end; 26 end; 27 end;</pre>	<pre> struct point { int inf; point *leg; }; typedef point* sir[100]; void vec_l(sir &a) { int x,i,j,e; point *p; scanf("%d",&n); for (i=1; i<=n; i++) { a[i]=NULL; scanf("%d",&x); for (j=2; x!=1; j++) { for (e=0;x%j==0;x/=j) e++; if (e>0) { p=new point; p->inf=j; p->leg=a[i]; a[i]=p; } } } }</pre>
---	--

1.1.3 Probleme propuse

1. Considerăm o listă simplu înlănțuită ce memorează valori întregi. Realizați un subprogram care primește la apel, prin intermediul unui parametru p , adresa de început a unei astfel de liste și efectuează transferul primelor două elemente la finalul ei. Nu se va folosi memorie suplimentară. Noua adresă de început a listei va fi returnată de subprogram prin intermediul parametrului p .

Exemplu: Dacă lista conținea inițial valorile 1, 2, 3, 4, 5, 6, în urma executării subprogramului, ea va reține elementele 3, 4, 5, 6, 1, 2.

2. Considerăm o listă simplu înlănțuită ce memorează valori întregi distincte. Realizați o funcție care primește la apel, prin intermediul unui parametru p , adresa de început a unei astfel de liste și returnează numărul de elemente memorate în listă înaintea celui de cheie minimă.

Exemplu: Dacă lista conține valorile 10, 21, 3, 4, 5, 6, se va afișa 2.

3. Considerăm două liste dublu înălțuite ce memorează valori întregi. Realizați un subprogram care concatenează cele două liste după regula: lista pentru care suma valorilor memorate este mai mare va fi concatenată la finalul celeilalte. Adresele de început ale celor două liste sunt primite de subprogram prin intermediul a doi parametri, iar adresa de început a listei nou obținute va fi returnată prin al treilea parametru.

Exemplu: Dacă cele două liste conțin inițial valorile 1, 2, 3, 4, respectiv 5, 6, 10, atunci lista obținută în urma concatenării conține 1, 2, 3, 4, 5, 6, 10.

4. Considerăm o listă simplu înălțuită ce memorează valori întregi. Realizați un subprogram care primește la apel, prin intermediul unui parametru p , adresa de început a unei astfel de liste și efectuează inserarea a două elemente de informație nulă înainte și după fiecare element ce reține informația maximă.

Exemplu: Dacă lista conține valorile 10, 21, 3, 21, 21, 6, la finalul executării subprogramului lista va memora valorile 10, 0, 21, 0, 3, 0, 21, 0, 0, 21, 0, 6.

5. Considerăm o listă simplu înălțuită ce memorează valori întregi. Realizați un subprogram care primește la apel, prin intermediul unui parametru p , adresa de început a unei astfel de liste și efectuează ștergerea elementelor ce rețin ca informație media aritmetică a vecinilor săi.

Exemplu: Dacă lista conține valorile 1, 2, 3, 4, 7, 7, 7, la finalul executării subprogramului lista va memora valorile 1, 4, 7, 7.

6. Considerăm o listă dublu înălțuită ce memorează un șir de valori întregi. Realizați un subprogram care primește la apel, prin intermediul parametrilor p și u , adresa primului și a ultimului element ale unei astfel de liste și efectuează ștergerea elementului din mijloc. Dacă lista conține un număr par de elemente, atunci se vor șterge cele două elemente situate în mijloc.

Exemplu: Dacă lista conține valorile 1, 2, 3, 4, 7, 7, la finalul executării subprogramului lista va memora valorile 1, 2, 7, 7.

7. Considerăm o listă simplu înălțuită ce memorează valori întregi. Realizați un subprogram care primește la apel, prin intermediul unui parametru p , adresa de început a unei astfel de liste și efectuează ștergerea elementelor cu număr de ordine par: al doilea, al patrulea, al șaselea, ș.a.m.d.

Exemplu: Dacă lista conține valorile 1, 5, 3, 4, 7, 6, 8, la finalul executării subprogramului lista va memora valorile 1, 3, 7, 8.

8. Considerăm două liste simplu înălțuite care rețin valori întregi ordonate crescător. Realizați un subprogram care interclasează două astfel de liste prin inversarea legăturilor elementelor. Subprogramul primește adresele de început ale listelor prin doi parametri și returnează tot printr-un parametru adresa listei obținute.

Exemplu: Dacă cele două liste conțin inițial valorile 1, 2, 8, 9, respectiv 5, 6, 10, atunci lista obținută în urma interclasării conține 1, 2, 5, 6, 8, 9, 10.

9. Considerăm o listă simplu înlănțuită ce memorează valori distincte întregi. Realizați un subprogram care primește la apel, prin intermediul unui parametru p , adresa de început a unei astfel de liste și efectuează ștergerea elementelor situate între cele care rețin valorile x , respectiv y . Aceste două valori întregi sunt primite de subprogram tot prin parametri.

Exemplu: Considerăm $x=3$, $y=6$ și lista 1, 5, 6, 4, 7, 3, 8. La finalul executării subprogramului lista va memora valorile 1, 5, 8.

10. Considerăm o listă circulară simplu înlănțuită ce memorează cifre din baza 10. Realizați o funcție care primește la apel, prin intermediul unui parametru p , adresa de început a unei astfel de liste și efectuează ștergerea primului și al ultimului element. Funcția va returna noua adresă de început a listei.

Exemplu: Dacă lista conținea cifrele 1, 5, 3, 4, 7, 6, 8, la finalul executării subprogramului lista va memora valorile 5, 3, 4, 7, 6.

11. Fie o listă simplu înlănțuită de numere întregi nenule. Realizați un subprogram, care primind la apel, prin intermediul unui parametru, adresa de început a unei astfel de liste, permite inserarea între oricare două noduri ce memorează valori de același semn, a unui nou element a cărui valoare este egală cu suma celor două.

Exemplu: Dacă lista primită memorează valorile 3, 1, -2, 2, 4, 7, -8, 1, atunci subprogramul va modifica lista după cum urmează 3, 4, 1, -2, 2, 6, 4, 11, 7, -8, 1.

12. Scrieți un subprogram care primește, prin intermediul unui parametru p , adresa de început a unei liste simplu înlănțuite și efectuează ștergerea tuturor nodurilor care conțin valori prime. Noua adresă de început va fi returnată de subprogram tot prin intermediul parametrului p .

Exemplu: Dacă lista primită memorează valorile 3, 1, 12, 2, 4, 7, 8, atunci subprogramul va modifica lista după cum urmează: 1, 12, 4, 8.

13. Se citesc de la tastatură valori naturale până la întâlnirea lui 0. Valorile citite sunt memorate într-o listă liniară simplu înlănțuită. Pentru o valoare x citită, să se separe nodurile în două noi liste, una care să conțină valorile mai mici decât x , iar cea de a doua, pe cele mai mari decât x .

Exemplu: Dacă $x=4$, iar lista inițială conține valorile 3, 7, 1, 2, 2, 4, 8, 0, atunci programul va separa valorile în două liste ce conțin numerele 3, 1, 2, 2, 0, respectiv 7, 4, 8.

14. Să se realizeze un program care creează o listă simplu înlănțuită de tip stivă ce memorează primele n numere palindroame de minimum două cifre și care permite afișarea acestora. Programul va conține următoarele subprograme:

- un subprogram care permite crearea stivei. Subprogramul va primi la apel printr-un parametru întreg valoarea n și va întoarce prin alt parametru adresa vârfului;

- un subprogram care permite afișarea valorii memorate în vârful stivei. Adresa acesteia este primită printr-un parametru;
- o funcție care efectuează extragerea nodului din vârful stivei a cărui adresă este primită printr-un parametru. Adresa noului vârf este returnată de funcție ca rezultat.

Exemplu: Dacă $n=5$ se va afișa 55, 44, 33, 22, 11.

15. Considerăm că în fișierul *in.txt* se află numere întregi separate în cadrul liniilor prin câte un spațiu. Să se creeze o listă simplu înlănțuită cu valorile preluate de pe prima linie din *in.txt*. Realizați un program în care sunt definite următoarele subprograme:

- o funcție care permite crearea listei. Funcția va întoarce adresa de început a listei;
- un subprogram care să permită inserarea după elementele cu cheia maximă, a altor două elemente ce memorează primii doi multipli ai acestei chei. Subprogramul va primi printr-un parametru adresa de început a listei.
- un subprogram care să permită ștergerea din listă a tuturor elementelor care memorează cheia minimă a listei. Subprogramul va primi prin parametrul *p*, adresa de început a listei și va returna tot prin parametrul *p*, noua adresă de început.

Exemplu: Considerăm că fișierul *in.txt* conține pe prima linie valorile 3, 7, 1, 2, 1, 4, 7. Programul va crea o listă cu aceste numere. Apelul la procedura de inserare va modifica lista astfel: 3, 7, 14, 21, 1, 2, 1, 4, 7, 14, 21. Apelul la procedura de ștergere va modifica lista inițială astfel: 3, 7, 2, 4, 7.

16. Considerăm o listă simplu înlănțuită ce memorează valori întregi. Realizați un subprogram care, primind la apel, prin intermediul unui parametru, adresa de început a unei astfel de liste, separă elementele în două liste, una formată din nodurile cu număr de ordine impar (primul, al treilea, ș.a.m.d), iar cealaltă listă, din nodurile cu număr de ordine par (al doilea, al patrulea, ș.a.m.d). Subprogramul va returna adresele de început ale celor două liste create prin intermediul a doi parametri.

Exemplu: Dacă lista primită conține valorile 3, 1, 2, 2, 4, 7, 8, 1, atunci subprogramul va separa nodurile în două liste 3, 2, 4, 8, respectiv 1, 2, 7, 1.

17. Considerăm o listă simplu înlănțuită ce memorează valori întregi. Realizați un program care identifică informația cu frecvență maximă de apariție. Dacă există mai multe asemenea valori se va afișa una singură.

Programul va conține următoarele subprograme:

- funcția recursivă *Nr*, care returnează numărul de apariții al unei valori într-o listă simplu înlănțuită. Valoarea și adresa de început a listei vor fi primite la apel prin intermediul a doi parametri.

- subprogramul *Print*, care permite afișarea informației cu frecvență maximă de apariție dintr-o listă simplu înlănțuită, folosind apeluri la funcția *Nr*. Subprogramul primește adresa de început a listei printr-un parametru.

Exemplu: Dacă lista primită conține valorile 3, 1, 2, 2, 4, 7, 2, 2 atunci se va afișa valoarea 2 deoarece apare de 4 ori.

18. Realizați un subprogram care să schimbe adresele de legătură ale elementelor unei liste simplu înlănțuite astfel încât acestea să fie ordonate crescător după valorile pe care le memorează.

19. Se citește de la tastatură un șir de valori naturale care se termină cu 0 (valoare ce nu aparține șirului). Aceste numere sunt memorate cu ajutorul unei liste simplu înlănțuite. Realizați un program care permite ștergerea din listă a oricăror două elemente consecutive, de informații egale. Programul va conține următoarele subprograme:

- funcția *Make*, care permite crearea listei și care returnează adresa de început a acesteia;
- funcția *Del*, care realizează ștergerea nodurilor cu proprietatea specificată în enunț. Adresa de început a listei este primită printr-un parametru. Funcția va returna adresa de început a listei după efectuarea ștergerii.
- subprogramul *Print*, care permite afișarea informațiilor memorate într-o listă simplu înlănțuită. Subprogramul primește adresa de început a listei printr-un parametru.

Exemplu: Dacă lista primită conține valorile 3, 3, 3, 2, 4, 7, 2, 2, la finalul executării programului se va afișa: 3, 2, 4, 7.

20. Construiți și afișați o listă liniară simplu înlănțuită alocată dinamic care să conțină primele n numere prime. Începând cu cel mai mic număr, să se șteargă elementele numărând din k în k . Informațiile elementelor șterse vor fi afișate în ordinea eliminării lor.

Exemplu: Pentru $n=5$ și $k=3$, se va afișa: 2 3 5 7 11. Numerele în ordinea ștergerii elementelor sunt: 5 2 11 3 7.

21. Să se creeze o listă simplu înlănțuită care să conțină numere naturale distincte, cu valori mai mici ca 200000. Citirea numerelor se încheie la întâlnirea lui 0, valoare care nu va face parte din listă. Realizați un program care să permită afișarea informațiilor situate între elementul de informație minimă și cel de informație maximă. Programul va conține următoarele subprograme:

- funcția *Make*, care permite crearea listei și care returnează adresa de început a acesteia.
- funcția *Elem* care permite returnarea adresei la care se găsește elementul de informație minimă sau maximă. Funcția primește prin parametrul *prim* adresa de început a listei și prin parametrul întreg *t* valoarea 1 sau -1, după cum se dorește returnarea adresei minimului, respectiv a maximului.

c) subprogramul *Print*, care permite afișarea informațiilor memorate într-o listă simplă înălțuită situate între două adrese primite prin doi parametri.

Exemplu: Dacă lista primită conține valorile 3, 13, 3, 2, 4, 7, 1, 12, la finalul executării programului se va afișa 13, 3, 2, 4, 7, 1.

22. Considerăm un șir de n numere naturale. Pentru fiecare valoare citită se dorește reținerea într-o listă simplă înălțuită a primilor n multipli ai săi. Realizați un subprogram care permite citirea celor n numere și care memorează, într-un tablou unidimensional, fiecare adresă de început a listelor create.

23. Considerăm două liste dublu înălțuite care rețin valori întregi și pentru care se cunosc adresele de început. Realizați un subprogram care verifică dacă lista transmisă prin primul parametru se poate obține din a doua (transmisă prin al doilea parametru) prin eliminarea unora dintre elementele acesteia. În urma executării subprogramului se va afișa pe ecran mesajul *DA* sau *NU*.

Exemplu: Dacă prima listă conține valorile 3, 13, 3, 2, 4, 7, iar a doua 3, 2, 7, la finalul executării programului se va afișa *DA*.

24. Să se creeze o listă dublu înălțuită cu caractere citite de la tastatură. Citirea se încheie tastând caracterul '.'. Pornind de la aceasta, să se creeze alte două liste care să conțină: una, literele listei inițiale, iar cealaltă, restul caracterelor. Cele două liste vor fi create prin schimbarea legăturilor listei inițiale (fără a folosi memorie suplimentară).

Exemplu: Dacă lista primită conține caracterele 'b', 'c', '*', '2', 'A', 'W', 'k', '2', atunci programul va separa caracterele în două liste ce rețin 'b', 'c', 'A', 'W', 'k', respectiv '*', '2', '2'.

25. Să se realizeze câte un subprogram pentru calculul sumei, respectiv a produsului a două polinoame memorate cu ajutorul listelor dublu înălțuite. Un element al listei reține coeficientul și gradul unui monom, respectiv adresele elementelor vecine lui (succesor-predecesor).

Exemplu: Presupunând că prima listă memorează polinomul X^3+2X+1 , atunci elementele ei vor reține valorile (1,3) (2,1) (1,0). Dacă a doua listă memorează polinomul $3X+13$, atunci elementele ei vor reține valorile (3,1) (13,0). Lista care corespunde sumei celor două polinoame va reține valorile (1,3) (5,1) (14,0).

26. Considerăm o listă simplă înălțuită ce memorează cifrele unui număr. Realizați un subprogram care construiește o listă simplă înălțuită circulară cu cifrele distincte din lista a cărei adresă de început o primește printr-un parametru.

Exemplu: Dacă lista primită memorează valorile 3, 1, 3, 2, 4, 7, 1, 2, subprogramul va permite crearea unei liste circulare în care vor fi reținute cifrele 3, 1, 2, 4, 7.

27. Se introduc de la tastatură valori reale până când valoarea citită reprezintă media aritmetică a ultimelor două valori introduse. Realizați un subprogram care permite memorarea acestora într-o listă dublu înălțuită în care valorile sunt reținute în ordine crescătoare. Nu se va efectua operația de sortare a valorilor din listă.

Exemplu: Dacă se citesc valorile 2.4, 23.1, 4.5, 39.2, 4.0, 3.0, 3.5, atunci lista va reține, în ordine, valorile 2.4, 3.0, 3.5, 4.0, 4.5, 23.1, 39.2.

28. Considerăm două mulțimi memorate cu ajutorul listelor simplu înlănțuite. Să se realizeze câte o funcție pentru fiecare dintre operațiile următoare: intersecția, reuniunea și diferența a două mulțimi. Funcțiile vor întoarce adrese de început ale listelor create. În cadrul acestor subprograme se va apela funcția *Apare*, care verifică dacă o valoare se află memorată într-o listă. Atât valoarea cât și adresa de început a listei vor fi primite de funcție prin intermediul a doi parametri. Funcția *Apare* returnează în Pascal valoarea *True* sau *False*, respectiv 0 sau 1 în C++.

Exemplu: Dacă prima listă conține valorile 3, 13, 23, 2, 4, 7, iar a doua 3, 2, 7, 19, 34, atunci vor fi create următoarele liste:

Intersecția: 3, 2, 7; Reuniunea: 3, 13, 23, 2, 4, 7, 19, 34; Diferența: 13, 23, 4.

29. Se citesc mai multe numere care sunt memorate într-o listă circulară simplu înlănțuită. Citirea se încheie când este introdus un număr egal cu suma precedentelor două. Să se realizeze un program care afișează valorile din listă în ordinea inversă introducerii. Programul va conține un subprogram recursiv care afișează elementele listei în ordinea cerută.

30. Realizați un program care calculează matricea sumă a două matrice rare memorate prin intermediul listelor simplu înlănțuite. Prin matrice rară se înțelege o matrice cu majoritatea elementelor nule. Fiecare element din listă reține linia, coloana și valoarea unui element nenul din matricea respectivă.

1.2. Arbori și arborescențe

1.2.1 Concepte teoretice fundamentale

Arbori

Arbore \Leftrightarrow graf neorientat conex și aciclic (fără cicluri).

Fie un graf neorientat G cu n vârfuri. Următoarele afirmații sunt echivalente:

G este arbore $\Leftrightarrow G$ are $n-1$ muchii și este conex

G este arbore $\Leftrightarrow G$ are $n-1$ muchii și nu conține cicluri

G este arbore \Leftrightarrow oricare pereche de noduri este unită printr-un singur lanț.

Arbori cu rădăcină \Leftrightarrow graf neorientat conex fără cicluri, în care unul dintre noduri este desemnat ca rădăcină. Nodurile pot fi așezate pe niveluri începând cu rădăcina, care este plasată pe primul nivel.