c; keresés a listában:

```
fgv listakeres(fej: mutatótíp; adat: értéktíp; hely: mutatótíp): logikai;
változó p: mutatótíp
p := fej;
amíg (p <> végjel) és (elem[p].érték <> adat) ism
    p := elem[p].köv;
a vége;
hely := p;
listakeres := (p <> végjel);
vége.
```

4; Műveletek a lista elején:

a; hozzáfűzés a lista elejére:

```
fgv listafűzelől(fej: mutatótíp; adat: értéktíp): logikai
változó p: mutatótíp;
ha lefoglal(p) akkor
    elem[p].érték := adat;
    elem[p].köv := fej;
    fej := p;
    listafűzelől := igaz
különben
    listafűzelől := hamis;
a vége;
vége.
```

b., Törlés a lista elejéről:

```
fgv listatörlelöl (fej: mutatótip): logikai;
  változó p: mutatótip;
  ha fej <> végjel akkor
    p := fej;
    fej := elem [fej].köv;
    felszabadít (p);
    listatörlelöl := igaz.
  különben
    listatörlelöl := hamis;
  ha vége;
  vége.
```

c., Hozzáfűzés adott elem mögé:

```
fgv listafűzmögé (emögé: mutatótip; adat: értéktip): logikai;
  változó p: mutatótip;
  ha (emögé <> végjel) és lefoglal (p) akkor
    elem [p].érték := adat;
    elem [p].köv := elem [emögé].köv;
    elem (emögé).köv := p;
    listafűzmögé := igaz
  különben
    listafűzmögé := hamis;
  ha vége;
  vége.
```

d; Törlés adott elem mögül:

```
fgv listatörölmögül (emögül : mutatótíp): logikai;
változó p: mutatótíp;
ha (emögül <> végjel) és (elem[emögül].köv <> végjel) akkor
  p := elem[emögül].köv;
  elem[emögül].köv := elem[p].köv;
  felszabadít(p);
  listatörölmögül := igaz
különben
  listatörölmögül := hamis;
ha vége;
vége.
```

5; Hátrelévők a lista végén:

a; Hozzáfűzés a lista végén:

```
fgv listafűzvég (fej: mutatótíp; adat: értéktíp): logikai;
változó p: mutatótíp;
ha fej = végjel akkor
  listafűzvég := listafűzelől(fej, adat)
különben
  p := fej;
  amíg elem[p].köv <> végjel ism
      p := elem[p].köv;
  a vége;
  listafűzvég := listafűzmögé(p, adat);
ha vége;
vége.
```

b; Törlés a lista végéről:

```
fgv listatörölvég (fej : mutatótíp): logikai;
  változó pe, p : mutatótíp;
ha fej <> végjel akkor
    ha elem[fej].köv = végjel akkor
      listatörölvég := listatörölelsőt(fej)
    különben
      pe := fej
      p := elem[fej].köv;
      amíg elem[p].köv <> végjel ism
        pe := p;
        p := elem[p].köv;
      a vége;
      listatörölvég := listatörölmögül(pe)
    h vége
  különben
    listatörölvég := hamis;
h vége;
vége.
```

6, Rendezett lista:

```
fgv listafűzrend ( fej : mutatótip; adat : értéktip): logikai;
változó pe, p : mutatótip;
ha (fej = vegjel) vagy ( adat < elem[fej].érték ) akkor
    listafűzrend := listafűzelöl ( fej , adat)
különben
  pe := fej;
  p := elem[fej].köv;
  amíg (p<> végjel) és ( elem[p].érték < adat) ism
      pe := p;
      p := elem[p].köv;
  a vége;
  listafűzrend := listafűzmögé (pe, adat);
ha vége;
vége.
```

b., Törlés rendezett listából:

```
fgv listatörlrend ( fej : mutatótíp; adat : értéktíp ): logikai;
változó pe, p : mutatótíp;
ha fej <> végjel akkor
    ha elem [fej]. érték = adat akkor
        listatörlrend := listatörlelöl (fej)
    különben
        pe := fej;
        p := elem [fej]. köv;
        amíg (p <> végjel) és (elem [p]. érték < adat) ismm
            pe := p;
            p := elem [p]. köv;
        a vége;
        ha (p <> végjel) és (elem [p]. érték = adat) akkor
            listatörlrend := listatörlmögül (pe)
        különben
            listatörlrend := hamis;
        h vége;
    h vége
különben
listatörlrend := hamis;
h vége;
vége.
```

7., Strázsás lista:

a., inicializálás:

fgv slistaüres (fej, vég : mutatótíp): logikai

ha üres(fej) akkor

vég := fej;

slistaüres := igaz

különben

slistaüres := hamis,

ü vége,

vége.

b., strázsás lista feldolgozása:

eljárás slistafeldolg (fej, vég : mutatótíp);

változó p : mutatótíp;

p := fej;

amíg p <> vég ism

elem[p]. érték feldolgozása

p := elem[p]. köv

a vége;

vége.

c; keresés rendezés listában

```
fgv listakeres (fej, vég : mutatótip ; adat : értéktip; ly :
                mutatótip) : logikai;
változó  p : mutatótip;
elem[vég].érték := adat;
p := fej;
amíg elem[p].érték <> adat ism
    p := elem[p].köv;
a vége;
ly := p;
listakeres := (p <> vég);
vége.
```

d; rendezés lista bővítése:

```
fgv listafűzés (elé, vég : mutatótip; adat : értéktip) : logikai;
változó  p : mutatótip
ha elfoglal(p) akkor
elem[p] := elem[elé];
elem[elé].érték := adat;
elem[elé].köv := p;
ha vég = elé akkor
    vég := p;
h vége
listafűzés := igaz
különben
  listafűzés := hamis;
h vége;
vége.
```

e; Elem törlése láncolás listából:

```
fgv listatörül (ezt, vég : mutatótíp): logikai;
változó p: mutatótíp;
ha ezt <> vég akkor
  p := elem[ezt].köv;
  elem[ezt] := elem[p];
  ha vég = p akkor
      vég := ezt;
  h vége;
  felszabadít(p);
  listatörül := igaz
különben
  listatörül := hamis;
h vége;
vége.
```

REKURZIÓ:

1, Megvalósítására:

```
eljárás Rek eljárás
fgv     fgv
...
Rek elj.
...
vége
```

2, Szubrutin:

```
eljárás Rek elj ...
fgv     fgv
ha határfeltétel akkor
  ...
  {visszalépés}
különben
  ...
  ... Rek elj fgv ...
    {rekurzív hívás}
  ...
h vége
vége
```

Egy szubrutint rekurzívnak mondunk, ha önmagára vonatkozó hívást tartalmaz.

3., Faktoriális:

```
fgv  fakt (n: egész): valós;
  ha  n = 0  akkor
    fakt := 1
  különben
    fakt := n * fakt (n-1);
  h vége;
  vége.
```

4., Fibonacci:

```
fgv fib (n: egész): valós,
  elágazás
    amikor  n = 0
      fib := 0;
    amikor  n = 1
      fib := 1
    különben
      fib := fib (n-1) * fib (n-2);
  e vége;
vége.
```

5., Hanoi tornyai:

```
eljárás Hanoi (db: egész; S1,S2,S3: Sorszter);
  ha db > 0 akkor
    hanoi (db-1, S1,S2,S3)
    ki: db, ". korong átrakása", S1, "->", S3;
    Hanoi (db-1, S2,S3,S1);
  h vége;
vége.
```

6; Gyorsrendezés (Quicksort)

```
eljárás gyorsrend (A : tömbtip ; felső, alsó : egész);
változó i,j : egész;
változó x, csere : elemtip;
i := alsó ; j := felső ;
x := A[(felső + alsó)/2]
ismétel
    amíg A[i] < x ismét
        i := i+1;
    a vége ;
    amíg A[j] > x ismét.
        j := j-1 ;
    a vége ;
    ha i < j akkor
        csere := A[i]; A[i] := A[j] ; A[j] := csere ;
    ha vége;
    ha i <= j akkor
        i := i+1 ; j := j-1;
    ha vége ;
i vége i > j esetén
ha alsó < j akkor
    gyorsrend (A, alsó, j);
ha vége ;
ha i < felső akkor
    gyorsrend (A, i, felső);
ha vége;
vége.
```

FA:

1, alapfogalmai:

fa ; új elem ; gyökér ; csomópont ; él, irányú ; út, út hossza ; szülő ; gyerek ; testvér ; levélelem ; bináris fa ; szigorúan bináris fa ;

2, bináris fa deklarációja:

```
konstans   maxelem =
konstans   végjel = ∅;
típus      mutatótip : egész;
típus      értéktip :
típus      binfaelemtip : record (
             érték : értéktip;
             bal, jobb : mutatótip
           )
változó    elem : tömb [1.. maxelem] binfaelemtip;
```

3, inicializálás

```
eljárás binfaread ( gyökér : mutatótip);
    gyökér := végjel;
vége.
```

4, Binfa bejárása:

a, preorder:

```
eljárás binfapreorder (p : mutatótip);
ha p <> végjel akkor
    elem[p].érték feldolgozása
    binfapreorder (elem[p].bal);
    binfapreorder (elem[p].jobb);
ha vége
vége.
```

b; inorder:

```
eljárás binfainorder( p:mutatótip);
ha p<> végjel akkor
    binfainorder (elem^.bal);
    elem^.érték feldolg.
    binfainorder (elem^.jobb);
 h vége;
vége.
```

c; postorder:

```
eljárás binfapostorder (p:mutatótip);
ha   p<> végjel akkor
    binfapostorder (elem^.bal);
    binfapostorder (elem^.jobb);
    elem^.érték feldolg.;
 h vége;
vége.
```

4; Bináris fa nemrekurzív bejárása:

a., preorder:

```
eljárás binfanemrepre ( p:mutatótip);
változó  verem :vermetip;
vált     p, f : mutatótip,
vált     vermes: logikai;
veremreset (verem);
vermes := verembe (verem, p)  és verembe (verem, 1)
amíg vermes és nem vheműres (verem) ism.
    vermes := veremből (f) és veremből (p);
    ha vermes és (p<> végjel) akkor
```

elágazás

amíg $f = 1$:

elem [p].érték feldolgozása

veremből := verembe(verem, p) és verembe(verem, 2);

veremből := verembe(verem, elem [p].bal) és verembe(verem, 1);

amíg $f = 2$:

veremből := verembe(verem, elem [p].jobb) és verembe(verem, 1);

e vége;

k vége;

a vége;

vége.

6, inorder:

```
eljárás infaneworder (p: mutatótip);
változó verem: veremtip;
változó p, f: mutatótip;
változó veremür: logikai;
veremreset (verem);
veremür := verembe (verem, p) és verembe (verem, 1)
amíg veremür és nem veremüres (verem) ismét
    veremür := veremből (f) és veremből (p)
    ha veremür és (p <> végjel) akkor
    elágazás
      akkor f = 1:
        veremür := verembe (verem, p) és verembe (verem, 2)
        veremür := verembe (verem, elem[p].bal) és verembe (verem, 1);
      akkor f = 2;
        elem[p].érték feldolgozása
        veremür := verembe (verem, elem[p].jobb) és
                   verembe (verem, 1)
    e vége;
  b vége;
a vége;
vége.
```

c.) postorder :

```
eljárás binfaelemekpost (p: mutatótíp);
változó verem: veremtíp;
változó p, f: mutatótíp;
változó vemenok: logikai;
veremkezd (verem);
vemenok := verembe (verem, p) és verembe (verem, 1);
amíg vemenok és nem veremüres (verem) ismét
    vemenok := veremből (f) és veremből (p);
    ha vemenok és (p<>végjel) akkor
        eldgazás
            amikor f=1;
            vemenok := verembe (verem, p) és verembe (verem, 2);
            vemenok := verembe (verem, elem[p].bal) és verembe (verem, 1);
            amikor f=2:
            vemenok := verembe (verem, p) és verembe (verem, 3)
            vemenok := verembe (verem, elem[p].jobb) és verembe (verem, 1)
            amikor f=3:
            elem[p].érték feldolg.
        e vége;
    h vége;
a vége;
vége.
```