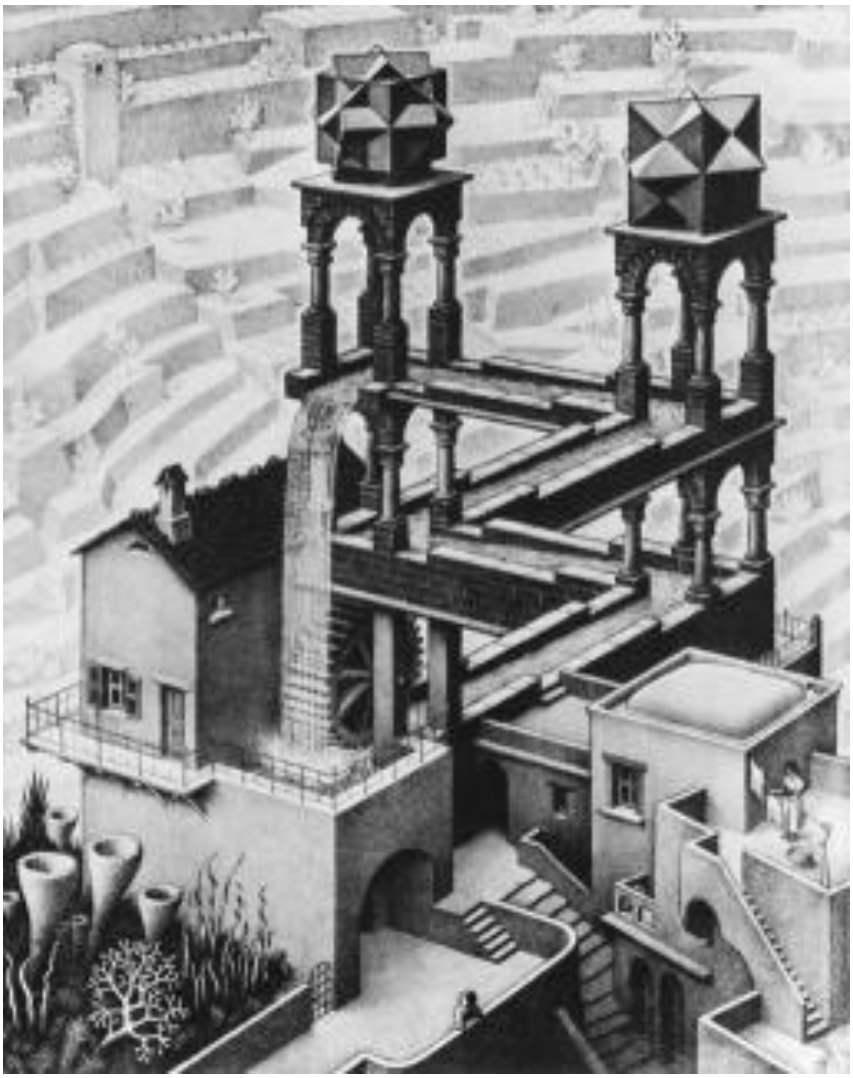
Rekurzió(k) - iteráció(k)

Szlávi Péter

2000

TARTALOM



ALAP REKURZÍV PÉLDÁK

Feladat: Az irodalomból ismert példák megvalósítása Pascalban figyelmünket két kérdésre összpontosítva: a rekurzív hívások száma, ill. a maximális veremmélység. Lássuk működés közben: REKURZIO.EXE!

```
Program Rekurzio;
  Uses Crt;
  Var  hivasSzam, aktMelyseg, maxMelyseg: LongInt;
       N,K: Word;
  Procedure Beolvas(Const kerdes:String;
                    Var egesz:Word; tol,ig:Integer);
  Var
    e:LongInt
  Begin
    If ig=-1 then ig:=MaxInt;
    Repeat
      Write(kerdes ,' [',tol,'..',ig,']:');
      {$i-}Readln(egesz);{$i+}
    Until (IOResult=0) and (egesz in [tol..ig]);
  End;
  Procedure Kiir(Const szoveg:String;
                 Const ertek:LongInt);
  Begin
    If ertek=-1 then {nincs érték!}
    Begin
      HighVideo; Writeln(#13#10,szoveg);
      NormVideo;
      Writeln('    hívásszám:',hivasSzam:8,',
        max.mélység:',maxMelyseg:8);
    End
    Else
    Begin
      HighVideo; Writeln(szoveg,ertek:8);
      NormVideo;
      Writeln('    hívásszám:',hivasSzam:8,',
        max.mélység:',maxMelyseg:8);
    End;
    ReadKey;
  End;
  Function Szov(Const ertek: Word):String;
    Var szoveg:String;
  Begin
    Str(ertek,Szoveg);
    Szov:=szoveg
  End;
```

```
{ ---------- Faktoriális sorozat ---------- }
  Function Fakt(Const n:LongInt):LongInt;
  Begin
    Inc(hivasSzam); Inc(aktMelyseg);
    If aktMelyseg>maxMelyseg then
      maxMelyseg:=aktMelyseg;
    If n=0 then Fakt:=1  else Fakt:=n*Fakt(n-1);
    Dec(aktMelyseg)
  End;
```

```
{ -------- Fibonacci sorozat --------------- }
  Function Fibo(Const n:Word):LongInt;
  Begin
    Inc(hivasSzam); Inc(aktMelyseg);
    If aktMelyseg>maxMelyseg then
      maxMelyseg:=aktMelyseg;
    If n in [0..1] then
       Fibo:=1
    else
       Fibo:=Fibo(n-1)+Fibo(n-2);
    {EndIf}
    Dec(aktMelyseg)
  End;
```

```
{ -------- Binomiális együtthatók ---------- }
  Function BinomP(Const n,k:Word):LongInt;
           //Pascal-∇
  Begin
    Inc(hivasSzam); Inc(aktMelyseg);
    If aktMelyseg>maxMelyseg then
      maxMelyseg:=aktMelyseg;
    If (n=0) or (k=0) or (n=k) then
          BinomP:=1
        else
          BinomP:=BinomP(n-1,k)+BinomP(n-1,k-1);
    {EndIf}
    Dec(aktMelyseg)
  End;
```

```
  Function BinomS(Const n,k:Word):LongInt;
           //Pascal-∇-ből egy Sor
  Begin
    Inc(hivasSzam); Inc(aktMelyseg);
    If aktMelyseg>maxMelyseg then
      maxMelyseg:=aktMelyseg;
    If (k>0) and (k<=n) then
      BinomS:=((n-k+1)*BinomS(n,k-1)) div k
    else
      BinomS:=1;
    {EndIf}
    Dec(aktMelyseg)
  End;
```

```
{ ------------- Hanoi tornyai -------------- }
Type   TPalcika = (Bal, Kozep, Jobb);
Const  SPalcika : Array [TPalcika] of
                  String[5]=
                  (' Bal ', 'Kozep', 'Jobb ')
```

```
Procedure Hanoi(Const rol,val,ra:TPalcika;
                Const n:Byte);
Begin
  Inc(hivasSzam);
  Inc(aktMelyseg);
  If aktMelyseg>maxMelyseg then
    maxMelyseg:=aktMelyseg;
  If n>0 then
  Begin
      Hanoi(rol,ra,val,n-1);
      Write(SPalcika[rol]+'->'+
            SPalcika[ra]+';':20);
      Hanoi(val,rol,ra,n-1);
  End;
  Dec(aktMelyseg)
End;
```

```
Begin
  ClrScr;  Writeln('Rekurziók':44);
  Beolvas('N:',N,0,12);
  aktMelyseg:=0; maxMelyseg:=0; hivasSzam:=0;
  Kiir('Fakt('+szov(N)+'):',Fakt(N));
  aktMelyseg:=0; maxMelyseg:=0; hivasSzam:=0;
  Kiir('Fibo('+szov(N)+'):',Fibo(N));
  Beolvas('K:',K,0,N);
  aktMelyseg:=0; maxMelyseg:=0; hivasSzam:=0;
  Kiir('BinomP('+szov(N)+','+szov(K)+'):',
    BinomP(N,K));
  aktMelyseg:=0; maxMelyseg:=0; hivasSzam:=0;
  Kiir('BinomS('+szov(N)+','+szov(K)+'):',
    BinomS(N,K));
  aktMelyseg:=0; maxMelyseg:=0; hivasSzam:=0;
  Hanoi(Bal,Kozep,Jobb,N);
  Kiir('Hanoi('+szov(N)+'):',-1{Nem érdekes});
End.
```

TÉTELEK REKURZÍV ÉS ITERATÍV MEGOLDÁSAI

Feladat: Az 'lineáris eldöntés' és '~ keresés', a 'logaritmikus eldöntés', ill. a 'maximum-kiválasztás', valamint az ezen alapuló rendezés' rekurzív és iteratív változatának összehasonlítása. Az összehasonlítás a tényleges végrehajtási idő és a ciklusok összvégrehajtási száma alapján történik. (A tényleges végrehajtási idő pontosabb mérése érdekében érdemes többször lefuttatni –természetesen– ugyanazon körülmények között, s ennek idejéből számítani az egyszeri végrehajtási időt.) Mielőtt elemezné, lássa működés közben: RekIter.exe!

Megoldás:

A megoldáshoz az alábbiakat érdemes fontolóra venni:

1) Az **ismétlési számok** megválasztása nyilvánvalóan függ az algoritmustól és az aktuális gép hardver-képességeitől. Ezért célszerű ezt **automatizálni.** Annál pontosabb a tényleges végrehajtási idő, minél többször hajtódik végre az adott alprogram, viszont a kivárhatóság is fontos szempont. Ezért jó döntésnek látszik az ismétlést addig végezni, amíg eltelik 1 másodperc. Ez alatt is horribilis számúszor hajtódik végre, úgy hogy ki kell használnunk a FreePascal nagy egész típusait. L. LongWord, ill. LongInt (és QWord) típusok tudnivalóit!

2) A rekurzív és iteratív algoritmusok összevetése csak akkor hiteles, ha garantáljuk a sorozatok elemeinek **azonos sorrendű feldolgozás**át. Ezért célszerű –és egyszerűbb– az iteratív algoritmusok ciklusait hátulról szervezni.

3) Ha egy rekurzív algoritmusban egy ágában többször is szerepel önmaga hívása (ilyen lesz a rendezésbeli maximum-index kiválasztása), akkor számítani kell a végrehajtási idők drasztikus növekedésére. Elemezze a rendezés algoritmusait ebből a szempontból is!

Először figyelje meg a keretprogrambeli típusdefiníciót, ill. -deklarációkat!

```
  …
Type
  Elem=LongInt;
  Tomb=Array [1..MaxN] of Elem;
Var
  N: LongWord;
  Y,X  : Tomb;            {sorozat/másolat-sorozat}
  mit  : Elem;            {Eldöntéshez/Keresésekhez}
  VanE : Boolean;         {Eldöntéshez/Keresésekhez}
  sorsz: LongWord;        {Kereséshez}
  RendetlensegiFok:Real;  {=Rendezetlenségi fok; 1 -> cc. 50%; 1/3 -> 33%}
  Nos  : Char;
  i,L,Li,Lr  : LongWord;  {ismétlési ciklushoz}
  IIdoTartam : LongWord;  {az iteratív időtartam, a rendezésnél}
  RIdoTartam : LongWord;  {a rekurzív időtartam, a rendezésnél}
  ICiklusSzam: LongWord;  {az iteratív ciklusszám, a rendezésnél}
  RCiklusSzam: LongWord;  {a irekurzív ciklusszám, a rendezésnél}
  IMIdoTartam: LongWord;  {a visszamásolási időtartam, az iteratív rendezésnél}
  RMIdoTartam: LongWord;  {a visszamásolási időtartam, a rekurzív rendezésnél}
  CiklusSzam : LongWord;
  invDb      : LongWord;  {inverziók száma, a rendezésnél}
  invDbS     : String;    {inverziók száma, karakteresen, a rendezésnél}
  rendFok    : Real;      {rendezetlenségi fok (%), a rendezésnél}
  rendFokS   : String;    {rendezetlenségi fok (%), karakteresen, a rendezésnél}
```

Vegy észre, hogy több programdarabból állítjuk össze a programot. Nézzen bele, hogy mi mit tartalmaz!

```
{$i idomeres.inc}  {$i egyebek.inc}
```

Az alábbi részben a vizsgált algoritmusok rekurzív és iteratív változatai következnek, Az iteratívak készek, a rekurzívak megoldása a feladat. Először készítse el ezek rekurzív specifikációit, majd implementálja! Figyelje meg az iteratívak ciklus-szervezését is!

```
Function EldontRek(Y: Elem; n:
         LongWord{; X: Tomb}): Boolean;
Begin
  Inc(CiklusSzam);
  {...idejön a lényeg...}
End; {EldontRek}
```

```
  Function EldontIte(Y: Elem; n: LongWord
                     {; X: Tomb}): Boolean;
    Var k: LongWord;
  Begin
    Inc(CiklusSzam);
    k:=n;
    While (k>0) and (Y<>X[k]) do
    Begin
      Dec(k);
      Inc(CiklusSzam);
    End;
    EldontIte:=k>0
  End; {EldontIte}
```

```
Procedure KeresRek(Y:Elem; n:LongWord{;X:Tomb};
          Var Van: Boolean; Var sorsz:LongWord);
Begin
  Inc(CiklusSzam);
  {...idejön a lényeg...}
End; {KeresRek}
```

```
Procedure KeresIte(Y:Elem; n:LongWord{;X:Tomb};
           Var Van:Boolean; Var sorsz:LongWord);
    Var k: LongWord;
  Begin
    Inc(CiklusSzam);
    k:=n;
    While (k>0) and (Y<>X[k]) do
    Begin
      Dec(k);
      Inc(CiklusSzam);
    End;
    Van:=k>0;
    If Van then sorsz:=k
  End; {KeresIte}
```

```
Function LogDontRek(Y:Elem; e,u:LongWord
                    {;X:Tomb}):Boolean;
  Var k: LongWord;
Begin
  Inc(CiklusSzam);
  {...idejön a lényeg...}
End; {LogDontRek}
```

```
Function LogDontIte(Y:Elem; e,u:LongWord
                    {; X: Tomb}):Boolean;
  Var k: LongWord;
Begin
  Inc(CiklusSzam);
  k:=(e+u) Div 2;
  While (u>=e) and (Y<>X[k]) do
  Begin
    Inc(CiklusSzam);
    If Y<X[k] then u:=k-1
              else e:=k+1;
    k:=(e+u) Div 2;
  End;{While};
  LogDontIte:=e<=u;
End; {LogDontIte}
```

```
Function MaxIndexRek(n:LongWord
                     {;X:Tomb}):LongWord;
Begin
  Inc(CiklusSzam);
  {...idejön a lényeg...}
End; {MaxIndexRek}
```

```
Function MaxIndexIte(n:LongWord
                     {;X:Tomb}):LongWord;
  Var maxi,i: LongWord;
Begin
  Inc(CiklusSzam);
  maxi:=n;
  For i:=n-1 downto 1 do
  Begin
    Inc(CiklusSzam);
    If X[i]>X[maxi] then maxi:=i
  End;
  MaxIndexIte:=maxi;
End; {MaxIndexIte}
```

```
Procedure MaxRendRek(n:LongWord{;X:Tomb});
  Var i: LongWord;
      seged: Elem;
Begin
  Inc(CiklusSzam);
  {...idejön a lényeg...}
End; {MaxRendRek}
```

```
Procedure MaxRendIte(n:LongWord{;X:Tomb});
  Var i,j,maxj: LongWord;
      seged: Elem;
Begin
  For i:=n downto 2 do
  Begin
    maxj:=i;
    For j:=i-1 downto 1 do
    Begin
      If X[maxj]<X[j] then maxj:=j;
      Inc(CiklusSzam);
    End;
    seged:=X[i]; X[i]:=X[maxj];
    X[maxj]:=seged;
  End;
End; {MaxRendIte}
```

Végül a főprogram következik, amely megszervezi magát a mérést. Először a sorozatot generálja le, amely –gondolva a logaritmikus eldöntésre– (növekvően) rendezett lesz.

```
  UjLap('Iteráció - rekurzió.');
  SorozatGeneralas;
  SorozatKi('A sorozat');
```

Ezt követik –azonos szerkezetben– a lineáris eldöntés, a lineáris keresés és a logaritmikus eldöntés hatékonyságmérő programdarabjai. Példaként elegendő az elsőt ide idézni:

```
…
    {                              Eldöntés                                  }
  Repeat
    UjLap('Lineáris eldöntés (N:'+NS+').');
    KeresettElem(mit);

    MeresIndulj;
    L:=1;
    While (ElteltSzMP<100) do
    Begin
      VanE:=EldontIte(mit,N{,X});
      Inc(L);
    End;
    MeresAllj;
    Writeln('Iterációval: ',VanE);
    Writeln('  Ideje:',IdoTartam/L:8:5,'    CiklusSzam:',CiklusSzam/L:6:0);
    MeresIndulj;
    L:=1;
    While (ElteltSzMP<100) do
    Begin
      VanE:=EldontRek(mit,N{,X});
      Inc(L);
    End;
    MeresAllj;
    Writeln('Rekurzióval: ',VanE);
    Writeln('  Ideje:',IdoTartam/L:8:5,'    CiklusSzam:',CiklusSzam/L:6:0);

    Varj(Nos);
  Until Nos=Space;
…
```

Mivel a maximum kiválasztásánál döntő, hogy hányadik lépésnél ér a maximumhoz, ezért mielőtt az eljáráspárt elindítanánk, célszerű némi keverést alkalmazni. Mivel a rekurzív kellően lassú tud lenni, ezért a keveredés fokát célszerű felhasználói döntéstől függővé tenni. Erre szolgál a `SorozatKever` eljárás.

A rendezésnél vannak extra meggondolni és tenni valók. Egy eleve rendezett sorozatot vagy nagyon gyorsan vagy –éppen ellenkezőleg– nagyon lassan, tehát nem tipikus idő alatt tud lerendezni. Ezért a kiinduló sorozatot előbb össze kell kissé keverni. A keveredés fokának megválasztását most is a felhasználó döntésére bízzuk.

Másik probléma az, hogy a rendezés után az ismétlődő rendezésnek is az eredeti rendezetlen sorozatot kell megkapnia. Ehhez szükséges egy másolat tömb, amelyben a kiinduló állapotú sorozat van. Ezt kell a rendezés megkezdése előtt a rendezésben szereplő tömbbe visszamásolni. És itt egy újabb probléma: ez a visszamásolás már számottevő időveszteséget okoz. Ráadásul a két változatban nem is egyforma arányban. Emiatt lényegesen meghamisítja az időarányokat. Ennek megoldása: le kell mérni, mennyi időt vett el az összidőből csak a másolás. Itt figyelnünk kell arra is, hogy a ciklusok szervezése ugyanolyan legyen!

Nézzük az iteratív változat mérésének megvalósítását! Ehhez hasonló a másik:

```
    {                  Maximum-kiválasztásos rendezés                       }
  Repeat
    UjLap('Maximum-kiválasztásos rendezések (N:'+NS+').');
    SorozatKever;
    Str(invDb,invDbS); rendFok:=200.0*invDb/N/(N-1); Str(rendFok:0:0,rendFokS);
    UjLap('Maximum-kiválasztásos rendezés (N:'+NS+',invSzám:'+invDbS+','+'rendFok:'+rendFokS+'%).');
    If N<=MaxKiN then {ellenőrzésképpen}
    Begin
      SorozatKi('A kevert sorozat');
      UjLap('Maximum-kiválasztásos rendezés (N:'+
            NS+',invSzám:'+invDbS+','+'rendFok:'+rendFokS+'%).');
    End;
```

```
    Write(#10#13+'Iteratív rendezés ... ':50);
    MeresIndulj;
    Li:=0;
    While (ElteltSzMP<200) {szándékosan 2-szerese a rekurzívnak:
                            így kisebb a másolási idő korrekciójának a hibája} do
    Begin
      X:=Y;   {rendezetlen állapot visszaállítása a rendezéshez}
      MaxRendIte(N{,X});
      Inc(Li);
    End;
    MeresAllj;
    Writeln('OK.');
    IIdoTartam:=IdoTartam;   {az iteratív számítás ideje}
    ICiklusSzam:=CiklusSzam; {az iteratív ciklusszám}
    If N<=MaxKiN then {ellenőrzésképpen}
    Begin
      SorozatKi('Eredmény -- Iteráció');
      UjLap('Maximum-kiválasztásos rendezés (N:'+
            NS+',invSzám:'+invDbS+','+'rendFok:'+rendFokS+'%).');
    End;
    //korrekció a másolási idővel: ---------------------------------------
    MeresIndulj;
    {A ciklus "bonyolítás" oka: az előbbi ciklushoz való nagyobb végrehajtási hasonlóság:}
    i:=1;
    While (i<=Li) and (-1<ElteltSzMP) do
    Begin
      X:=Y;   {eredeti helyzet visszaállítása}
      Inc(i);
    End;
    MeresAllj;
    IMIdoTartam:=IdoTartam; {a visszamásolás ideje}
    //a korrekció vége ----------------------------------------------
    …
```

A végén persze a mérések eredményeit meg kell jeleníteni:

```
    …
    Writeln(#10#13+'Iterációval: ');
    Writeln('  Ideje:',(IIdoTartam-IMIdoTartam)/Li:8:5,
            '    CiklusSzam:',ICiklusSzam/Li:6:0);
    Writeln('Rekurzióval: ');
    Writeln('  Ideje:',(RIdoTartam-RMIdoTartam)/Lr:8:5,
            '    CiklusSzam:',RCiklusSzam/Lr:6:0);
    Varj(Nos);
  Until Nos=Space;
```

Az ‘Időmérés.inc’ betét

```
Const
  Nagy0:LongWord=0;
Var
  KezdoIdo,VegIdo: LongInt;
  IdoTartam: QWord;
Procedure MeresIndulj;
  Var  o,p,mp,szmp: Word;
Begin
  GetTime(o,p,mp,szmp);
  KezdoIdo:=Nagy0+szmp+100*(mp+60*(p+60*o));
  CiklusSzam:=0;
End; {MeresIndulj}
```

```
Procedure MeresAllj;
  Var  o,p,mp,szmp: Word;
Begin
  GetTime(o,p,mp,szmp);
  VegIdo:=Nagy0+szmp+100*(mp+60*(p+60*o));
  IdoTartam:=VegIdo-KezdoIdo;
End; {MeresAllj}


Function ElteltSzMP:LongWord;
  Var o,p,mp,szmp: Word;
Begin
  GetTime(o,p,mp,szmp);
  ElteltSzMP:=Nagy0+szmp+
              100*(mp+60*(p+60*o))-KezdoIdo;
End;
```

Az ‘Egyebek.inc’ betét

```
Procedure UjLap(s: String);
Begin
  ClrScr;
  HighVideo;
  Writeln(s:40+(Length(s) Div 2));
  NormVideo;
End; {UjLap}
```

```
Procedure Varj(Var nos: Char);
Begin
  GotoXY(35,25);
  HighVideo;
  Write('Space-re kilép.');
  Nos:=ReadKey; NormVideo;
End;
```

```
Procedure SorozatGeneralas{(Var X: Tomb)};
  Var i: LongInt;
Begin
  Repeat
    Write(CrLf+'Sorozathossz:'); Readln(N);
  Until (N>10) and (N<=MaxN);
  Str(N,NS);
  X[1]:=Random(N); {kiinduló érték}
  Y[1]:=X[1];
  For i:=2 to N do
  Begin
   {legalább 2-vel odébb:}
    X[i]:=X[i-1]+Random(10)+2;
    Y[i]:=X[i];
  End;
End; {Sorozat}
```

```
Procedure SorozatKi(cim:String{;Var X:Tomb});
  Var i: LongInt;
Begin
  Writeln(CrLf,cim:(40+Length(cim) Div 2));
  {ha MaxKiN-nál kevesebb, kiírjuk:}
  If N<=MaxKiN then
  Begin
    For i:=1 to N do
    Begin
      Write(i:6,'.:');
      HighVideo; Write(X[i]:8);
      NormVideo;
    End;
    GotoXY(35,25);
    Write('Space-re tovább.');
    Nos:=ReadKey;
  End;
End; {SorozatKi}
```

```
Procedure SorozatKever{Var N:LongInt;X:Tomb};
  Var hiba,
      i,j: LongInt;
      e  : Elem;
      rS :String;
Begin
  Repeat
    Write(CrLf+'Rendezetlenségi fok (0..1;
              ajánlott:',1/N:5:4,'):');
    Readln(rS);
    If rS='' then
    Begin
      RendetlensegiFok:=1/N; hiba:=0;
    End
      else
    Begin
      Val(rS,RendetlensegiFok,hiba);
    End;
  Until (hiba=0) and (RendetlensegiFok<=1) and
                     (RendetlensegiFok>=0);
  Write(CrLf+'Keverés...');
  For i:=1 to N-1 do
  Begin
    If Random<RendetlensegiFok then
    Begin
      j:=Random(N-i+1)+i;
      e:=X[i]; X[i]:=X[j]; ; X[j]:=e;
    End;
  End;
  //inverziók száma:
  invDb:=0;
  For i:=1 to N-1 do
  Begin
    For j:=i+1 to N do
    Begin
      If X[i]>X[j] then Inc(invDb);
    End;
  End;
  Y:=X;
  Writeln('OK.');
End; {SorozatKever}
```

```
Procedure KeresettElem(Var mit: Elem);
  Var  i: LongInt;
Begin
  Writeln;
  Write('A keresettet megtaláljuk?');
  Nos:=ReadKey; Write(Nos);
  i:=Random(N Div 3)+1;
  If UpCase(Nos)='I' then
  Begin
    mit:=X[i];
    Writeln('  Mit:',mit,', sorszáma:',i,'.');
  End
      else
  Begin
    mit:=X[i]+1;
    Writeln('  Mit:',mit,'=X(',i,')+1');
  End;
End; {KeresettElem}
```

Nézzük végül a megoldásokat:

```
Function EldontRek(Y:Elem; n:LongWord
                  {; X:Tomb}):Boolean;
Begin
  Inc(CiklusSzam);
  If
      n=0   then EldontRek:=False else if
     Y=X[n] then EldontRek:=True
         else    EldontRek:=EldontRek(Y,n-1)
  {EndIf};
End; {EldontRek}
```

```
Procedure KeresRek(Y:Elem; n:LongWord {; X:Tomb};
           Var Van:Boolean; Var sorsz:Integer);
Begin
  Inc(CiklusSzam);
  If
    n=0   then
          Begin
            Van:=False; {sorsz:=Barmi}
          End else if
    Y=X[n] then
          Begin
            Van:=True;   sorsz:=n
          End
     else Begin KeresRek(Y,n-1,Van,sorsz); End
  {EndIf};
End; {KeresRek}
```

```
Function LogDontRek(Y:Elem; e,u:LongWord
                  {;X:Tomb}):Boolean;
   Var k: Integer;
Begin
  Inc(CiklusSzam);  k:=(e+u) Div 2;
  If
     u<e   then LogDontRek:=False else if
    Y=X[k] then LogDontRek:=True  else if
    Y<X[k] then LogDontRek:=LogDontRek(Y,e,k-1)
      else      LogDontRek:=LogDontRek(Y,k+1,u)
  {EndIf};
End; {LogDontRek}
```

```
Function MaxIndexRek(n:LongWord{;X:Tomb}):LongWord;
Begin
  Inc(CiklusSzam);
  If
    n=1 then
      MaxIndexRek:=n else if
    X[n]>X[MaxIndexRek(n-1{,X})] then
      MaxIndexRek:=n
    else
      MaxIndexRek:=MaxIndexRek(n-1{,X})
  {EndIf};
End; {MaxIndexRek}
```

```
Procedure MaxRendRek(n:LongWord{;X:Tomb});
  Var  i: LongWord;
       seged: Elem;
Begin
  If n>1 then
  Begin
    i:=MaxIndex(n{,X});
    seged:=X[n];  X[n]:=X[i];  X[i]:=seged;
    MaxRendRek(n-1{,X})
  End;
End; {MaxRendRek}
```

BACKTRACK

Feladat: A 'Backtrack' rekurzív átiratának elkészítése. Az unos-untig használt sablon a következő (Keresés) tétel átalakítása alapján könnyen megérthető:

Iteratív

```
Eljárás Keres(Konstans x:TSorozat;
              Változó Van:Logikai,melyik:Egész):
  Változó i:Egész
  i:=1
  Ciklus amíg i≤N és nem T(x,i)
    i:+1
  Ciklus vége
  Van:=i≤N
  Ha Van akkor melyik:=i
Eljárás vége.
```

Rekurzív

A ciklust jobb rekurzióvá alakíthatjuk, miután különválasztottuk (egy önálló eljárásba) az „előkétől" és az „utókától". Az eredeti eljárás paraméterei közül elegendő azokat „átvinni" a rekurzív eljárás paraméterei közé, amelyektől a ciklus valójában függ. Gondoskodni kell arról, hogy az utóka is a megfelelő információt megkapja. Erre sok mód lehet. Most a legkézenfekvőbbet válasszuk: legyen a rekurzív tevékenység egy index értékű függvény.

```
Eljárás Keres(Konstans x:TSorozat;
              Változó Van:Logikai,melyik:Egész):
  Változó i:Egész
  i:=1
  i:=KeresR(i,x)[1]
  Van:=i≤N
  Ha Van akkor melyik:=i
Eljárás vége.
```

A ciklus jobb rekurzióvá alakítása:

```
Függvény KeresR(Konstans i:Egész,
                        x:TSorozat):Egész
  Elágazás
    i>N          esetén KeresR:=i
    nem T(x,i)   esetén KeresR:=KeresR(i+1,x)
    egyéb        esetben KeresR:=i
  Elágazás vége
Függvény vége.
```

Alábbiakban a backtrack iskolapéldáját demonstráló programot adjuk meg iteratívan és rekurzívan. Mielőtt elemezné, lássa működés közben: VEZERDEMO.EXE. (Az azonos részeket csak egyszer adjuk meg.)

Iteratív

```
Program N_Vezer{Iteratív backtrack};
  Uses
     Newdelay,Crt,Graph,CursUnit;
  Const
     MaxN  = 10;
  Type
     Index = 0..MaxN+1;
     MIndex= -1..MaxN+1;
     Tabla = Array [1..MaxN] of (Ures,Foglalt);
     Hol   = Array [1..MaxN] of Index;
  Var
     Vezer : Hol;
     i: MIndex;
     N,j,
     hova  : Index;
     lehet : Boolean;
   {$I Vezer.inc}

  Function Uti(vx1,vy1, vx2,vy2: Index): Boolean;
  Begin
    Uti:=(vy1=vy2) or (Abs(vy1-vy2)=vx1-vx2)
  End; {Uti}

  Function RosszEset(x,y: Index): Boolean;
    Var
       j: Index;
  Begin
    Letesz(x,y,True);
    j:=1;
```

Rekurzív

```
Function RosszEset(x,y: Index): Boolean;
  Function RosszEsetR(j,x,y: Index): Boolean;
  Begin
    If
      j>x-1 then
           Begin
```

[1] Egyszerűbben is írható: ~~i:=1~~ i:=KeresR(1,x)

```
    While (j<=x-1)
          and not uti(x,y, j,Vezer[j]) do
    Begin
      Inc(j);
    End;
    Varakozas(1); Folvesz(x,y);
    RosszEset:=j<=x-1
  End;
```

```
  Procedure VanJoEset(i: Index;
            Var talan: Boolean; Var ide: Index);
  Begin
    ide:=Vezer[i]+1;
    While (ide<=N) and RosszEset(i,ide) do
      Inc(ide);
    talan:=ide<=N;
  End;
```

```
  Procedure BackTrack;
  Begin
    i:=1;
    For j:=1 to N do Vezer[j]:=0;
    While i in [1..N] do
    Begin
      Varakozas(2);
      VanJoEset(i,lehet,hova);
      If lehet then
      Begin
        Letesz(i,hova,False);
        Vezer[i]:=hova; Inc(i);
      End
        else
      Begin
        Vezer[i]:=0; Dec(i);
        If i>0 then
        Begin
          Folvesz(i,Vezer[i]);
        End;
      End;
    End;
    If i>0 then VanMegoldas(N)
           else NincsMegoldas(N);
  End;
```

```
Begin
  Inicializalas(N);
  CurOff;
  BackTrack;
  CurOn;
End.
```

```
            RosszEsetR:=False
          End else If
      not uti(x,y, j,Vezer[j]) then
            RosszEsetR:=RosszEsetR(j+1,x,y)
      else RosszEsetR:=True
    {EndIf};
  End;
Begin
  Letesz(x,y,True);
  RosszEset:=RosszEsetR(1,x,y);
  {az adminisztráció nélküli,
   rekurzív rész meghívása}
  Varakozas(1); Folvesz(x,y);
End;
```

```
Procedure VanJoEset(i: Index;
          Var talan: Boolean; Var ide: Index);
  Procedure VanJoEsetR(i: Index;
          Var talan: Boolean; Var ide: Index);
  Begin
    If
      ide>N  then
            talan:=False else if
      RosszEset(i,ide) then
            Begin
              Inc(ide);
              VanJoEsetR(i,talan,ide)
            End
      else  talan:=True;
    {EndIf}
  End;
Begin
  ide:=Vezer[i]+1;
  VanJoEsetR(i,talan,ide);
End;
```

```
Procedure BackTrack;
  Function BackTrackR(i:MIndex):Boolean;
  Begin
    If
      i<1 then BackTrackR:=False else if
      i>N then BackTrackR:=True
      else     Begin
                 Varakozas(2);
                 VanJoEset(i,lehet,hova);
                 If lehet then
                 Begin
                   Letesz(i,hova,False);
                   Vezer[i]:=hova; Inc(i);
                 End
                   else
                 Begin
                   Vezer[i]:=0; Dec(i);
                   If i>0 then
                   Begin
                     Folvesz(i,Vezer[i]);
                   End;
                 End;
                 BackTrackR:=BackTrackR(i);
               End;
    {EndIf};
  End;
Begin
  i:=1;
  For j:=1 to N do Vezer[j]:=0;
  If BackTrackR(i) then VanMegoldas(N)
                   else NincsMegoldas(N);
End;
```

A ‘Vezer.inc’ betét

```
  { megjelenítési paraméterek: }
Var
  bfx,bfy,
  jax,jay: Integer; {táblasarkak}
  mOldal : Integer; {egy mezô oldalának hossza}

Procedure Varakozas(meddig: Integer);
  Var
    c: Char;
Begin
  If meddig<=0 then
  Begin
    Repeat Until KeyPressed; c:=ReadKey;
  End
    else
  Begin
    Delay(meddig*1000);
    If KeyPressed then
    Begin
      c:=ReadKey;
      Repeat Delay(1000); Until KeyPressed;
      c:=ReadKey;
    End;
  End;
End; {Várakozás}

Procedure MezoRajz(x,y: Integer; szin: Integer);
Begin
  Window(bfx+2*mOldal*(x-1),bfy+mOldal*(y-1),bfx+2*mOldal*x-1,bfy+mOldal*y-1);
  TextBackGround(szin); ClrScr;
End; {MezôRajz}

Procedure BabuRajz(x,y: Integer; hszin,bszin: Integer);
  Var
    balx,baly: Integer;
Begin
  balx:=bfx+2*mOldal*(x-1); baly:=bfy+mOldal*(y-1);
  Window(balx,baly,balx+2*mOldal-1,baly+mOldal-1);
  TextBackGround(hszin); TextColor(bszin);
  GotoXY(mOldal,(mOldal Div 2)+1); Write('_'{=2});
End; {BábuRajz}

Procedure Inicializalas(Var n: Index);
  Var
    i,j: Integer;
Begin
  Window(1,1, 80,25); TextColor(White); TextBackGround(Black); ClrScr;
  Writeln('N-vezérprobléma':48);
  Repeat
    GotoXY(1,5); Write('Vezérszám (1<N<',MaxN:2,'):'); Readln(n)
  Until n in [2..MaxN];
  Window(1,2, 80,25); ClrScr;
  mOldal:=22 Div n;
  bfy:=2+(22 Div n) Div 2;    jay:=bfy+n*mOldal-1;
  bfx:=(80-2*n*mOldal) Div 2; jax:=bfx+2*n*mOldal;
  Window(bfx-1,bfy-1, jax+1,jay+1); TextColor(White); TextBackGround(Brown); ClrScr;
  Window(1,1, 80,25); {Ablakvisszaállítás}
  For i:=1 to n do For j:=1 to n do
  Begin
    If ((i+j) Mod 2)=0 then MezoRajz(i,j,White) else MezoRajz(i,j,Black);
  End
End;

Procedure Letesz(i,j: Index; proba:Boolean);
Begin
  If proba then
    If ((i+j) Mod 2)=0 then BabuRajz(i,j,White,Red)
                       else BabuRajz(i,j,Black,Yellow)
  Else
    If ((i+j) Mod 2)=0 then BabuRajz(i,j,White,Black)
                       else BabuRajz(i,j,Black,White);
End; {Letesz}
```

```
  Procedure Folvesz(i,j: Index);
  Begin
    If ((i+j) Mod 2)=0 then MezoRajz(i,j,White) else MezoRajz(i,j,Black);
  End; {Folvesz}

  Procedure VanMegoldas(n: Index);
    Var
      i,j: Index;
  Begin
    Window(1,1, 80,25); TextColor(White); TextBackGround(Black); ClrScr;
    Write('Az N-vezérproblémának N=',n,' esetre van megoldása, mégpedig az alábbi:');
    Window(bfx-1,bfy-1, jax+1,jay+1); TextColor(White); TextBackGround(Brown); ClrScr;
    Window(1,1, 80,25); {Ablakvisszaállítás}
    For i:=1 to n do For j:=1 to n do
    Begin
      If ((i+j) Mod 2)=0 then
      Begin
        MezoRajz(i,j,White);
        If Vezer[i]=j then BabuRajz(i,j,White,Black);
      End
        else
      Begin
        MezoRajz(i,j,Black);
        If Vezer[i]=j then BabuRajz(i,j,Black,White);
      End;
    End;
    Varakozas(0);
  End;

  Procedure NincsMegoldas(n: Index);
  Begin
    Window(1,1, 80,25); TextColor(White); TextBackGround(Black); ClrScr;
    Write('Az N-vezérproblémának N=',n,' esetre nincs megoldása.');
    Varakozas(0);
  End;
```

A ‘CursUnit.pas’ unit

```
Unit CursUnit;

Interface

  Procedure CurOn;
  Procedure CurOff;

Implementation
  Var Cursor : Word;

  Procedure CurOn; Assembler;
  Asm
    mov ah,01h
    mov cx,Cursor
    int 10h
  End; {CurOn}

  Procedure CurOff; Assembler;
  Asm
    mov ah,03h
    mov bh,00h
    int 10h
    mov Cursor, cx
    mov ah,01h
    mov cx,65535
    int 10h
  End; {CurOff}
Begin
End.
```

QUICKSORT

Feladat: A 'QuickSort rendezés' „alap" rekurzív (amelyben a 'Szétválogatás' iteratívan szerepelhet; REKQS.EXE), a fél-rekurzív (QS_FELRE.EXE) és az iteratív (ITERQS.EXE) változatának összevetése. Először a –közismert– rekurzív változatot tárgyaljuk!

```
Program FelRekurzivQuickSort;
  Uses  Newdelay,Crt;
  Const
    N = 100;
  Type
    Index=1..N;     IndexM=0..N+1;
    Intervallum=Record eleje,vege:IndexM End;
  Var
    tomb  : Array [Index] of Integer;
    szakasz,masikszakasz : Intervallum;
    kozep : IndexM;
  Const
    KezdoSzakasz:Intervallum=(eleje:1; vege:N);
  Procedure TombInic;
    Var  i : Integer;  c : Char;
  Begin
    LowVideo; Window(16,2,55,15);
    Writeln('A rendezetlen vektor:');
    For i:=1 to N do
    Begin
      tomb[i]:=Random(N);  Write(tomb[i]:4)
    End;
    c:=ReadKey;
  End; {TombInic}
  Procedure TombKiir;
    Var  i : Integer;   c : Char;
  Begin
    HighVideo; Window(16,14,55,25);
    Writeln('A rendezett vektor:');
    For i:=1 to N do
    Begin
      Write(tomb[i]:4)
    End;
    c:=ReadKey;
  End; {TombKiir}
```

```
  Function Uresszakasz(sz:Intervallum):Boolean;
  Begin
    UresSzakasz:=(sz.vege-sz.eleje)<1
  End; {UresSzakasz}
  Procedure Szetvalogat(sz:Intervallum;
                        Var k:Index);
    Var  seged : Integer;   bal,jobb  : IndexM;
  Begin
    bal:=sz.eleje; jobb:=sz.vege;
    seged:=tomb[bal];
    While bal<jobb do
    Begin
     While (bal<jobb) and (tomb[jobb]>=seged) do
       Dec(jobb);
     If bal<jobb then
     Begin
       tomb[bal]:=tomb[jobb]; Inc(bal)
     End;
     While (bal<jobb) and (tomb[bal]<=seged) do
       Inc(bal);
     If bal<jobb then
     Begin
       tomb[jobb]:=tomb[bal]; Dec(jobb)
     End;
    End;
    tomb[bal]:=seged;
    k:=bal
  End; {Szetvalogat}
```

```
  Procedure rQS(sz: Intervallum);
    Var kozep:Index;   szM:Intervallum;
  Begin
    Szetvalogat(sz,kozep);
    szM.eleje:=sz.eleje; szM.vege:=kozep-1;
    If not UresSzakasz(szM) then rQS(szM);
    szM.eleje:=kozep+1; szM.vege:=sz.vege;
    If not UresSzakasz(szM) then rQS(szM);
  End;
```

```
Begin
  ClrScr;
  TombInic; szakasz:=KezdoSzakasz;
  rQS(szakasz);
  TombKiir
End.
```

Másodikként a félig rekurzív megoldást nézzük meg. Az átírás lényegét az alábbi sémával közelíthetjük meg:

Az absztrakt rekurzív eljárás:

```
Eljárás A(K.x:T):
  y:=f(x)
  Ha p(y) akkor A(y)
  y:=g(x)
  Ha q(y) akkor A(y)
Eljárás vége.
```

['K.' ≡ 'Konstans'
f,g,p x-*invariáns*]

Vezessük be az alábbi eljárást:

```
Eljárás B(K.x:TX,V.y:T):
  y:=f(x)
  Ha p(y) akkor A(y)
  y:=g(x)
Eljárás vége.
```

['V.' ≡ 'Változó']

Vagyis az A eljárás B-vel rövidített alakban:

```
Eljárás A(K.x:T):
  B(x,y)
  Ha q(y) akkor A(y)
Eljárás vége.
```

Így az ekvivalens megoldás:

```
Eljárás A(K.x:T):
  B(x,y)
  Ciklus amíg q(y)
    B(y,y[új érték!])
  Ciklus vége
Eljárás vége.
```

A fenti „sémát" alkalmazva, s elvégezve a pontosítást, kapjuk az alábbit:

```
Procedure FelrekQS(sz: Intervallum);
  Var kozep: Index;
      szM  : Intervallum;
Begin
  Szetvalogat(sz,kozep);
  szM.eleje:=sz.eleje; szM.vege:=kozep-1;
  If not UresSzakasz(szM) then FelrekQS(szM);
  szM.eleje:=kozep+1; szM.vege:=sz.vege;
```

}

```
y:=f(x)
Ha p(y) akkor A(y)
y:=g(x)
```

⇒

```
x   y
↑   ↑
B(sz,szM)
```

```
  While not UresSzakasz(szM) do
  Begin
    Szetvalogat(szM,kozep);
    szM.eleje:=szM.eleje; szM.vege:=kozep-1;
    If not UresSzakasz(szM) then FelrekQS(szM);        B(szM,szM)
    szM.eleje:=kozep+1; szM.vege:=sz.vege;
  End;
End; {FelrekQS}
```

Harmadikként az iteratív megoldást nézzük meg. Ennek „alappillére” a veremhasználat. (Csak a „változó” részeket írjuk le.)

```
Program IterativQuickSort;
  Uses
   Crt;
  Const
    N = 100; VMax = 100;
  Type
    Index=1..N; Mutato=0 {=sehova}..N+1{=sehova};
    Intervallum=Record eleje, vege:Mutato End;
  Var
    verem:Array [1..VMax] of Intervallum;
    vm : Mutato;  {veremmutató}
    tomb: Array [Index] of Integer;
  Const
    KezdoSzakasz:Intervallum=(eleje:1; vege:N);
  Procedure Verembe(mit: Intervallum);
  Begin
    Inc(vm); verem[vm]:=mit
  End; {verembe}
  Procedure Verembol(Var  mibe: Intervallum);
  Begin
    mibe:=verem[vm]; Dec(vm)
  End; {Verembol}
  Procedure VeremInic;
  Begin
    vm:=0
  End; {VeremInic}
  Function UresVerem: Boolean;
  Begin
    UresVerem:=(vm=0)
  End; {UresVerem}
```

Az alábbi eljárások nem változnak:

```
  Procedure TombInic;          Function Uresszakasz(sz: Intervallum): Boolean;
  Procedure TombKiir;          Procedure Szetvalogat(sz: Intervallum; Var k: Index);
```

A lényeg:

```
Procedure iterQS(sz: Intervallum);
  Var  masikszakasz : Intervallum;   kozep : Index;
Begin
  {a híváskor megtörtént: TombInic;  szakasz:=KezdoSzakasz;}
  VeremInic;  Verembe(szakasz);
  While not UresVerem do
  Begin
    While not UresSzakasz(szakasz) do
    Begin
      Szetvalogat(szakasz,kozep);
      masikszakasz.eleje:=kozep+1; masikszakasz.vege:=szakasz.vege;
      Verembe(masikszakasz);
      szakasz.vege:=kozep-1
    End;
    Verembol(szakasz)
  End;  TombKiir
End {iterQS}.
```

KOCH FRAKTÁL

Feladat: A 'Koch-fraktál' iteratív és rekurzív megvalósítása Turbo Pascal-ban. Mielőtt elemezné, lássa működés közben: KOCH_R_I.EXE!

```
Program KochFraktal; { Rekurzívan és kétféleképpen iteratívan }
  Uses Graph3;
  Var  s,h,k,sz: Integer;
  Procedure Inic;
  Begin
    HiRes; HiresColor(15);
    Home; PenUp; TurnLeft(90); Forwd(150); TurnRight(180); PenDown;
  End;
```

```
  Procedure Koch(szint,hossz:integer);
  Begin
    If szint=0 then
    Begin
      Forwd(hossz)
    End
       Else
    Begin
      Koch(szint-1,hossz Div 3);
      TurnLeft(60);
      Koch(szint-1,hossz Div 3);
      TurnRight(120);
      Koch(szint-1,hossz Div 3);
      TurnLeft(60);
      Koch(szint-1,hossz Div 3);
    End;
  End;
```

```
  Const
    MaxMelyseg = 100;
  Var
    verem: Array [1..MaxMelyseg] of Record szint,kov,hossz,szog:Integer End;
    vm   : 0..MaxMelyseg;
```

```
  Procedure VeremInic;
  Begin
    vm:=0;
  End;
  Procedure Verembol(Var szint,kov,
                         hossz,szog: Integer);
  Begin
    If vm>0 then
    Begin
      szint:=verem[vm].szint;
      kov:=verem[vm].kov;
      hossz:=verem[vm].hossz;
      szog:=verem[vm].szog;
      Dec(vm);
    End
  End;
```

```
  Procedure Verembe(szint,kov,
                    hossz,szog: Integer);
  Begin
    If vm<MaxMelyseg then
    Begin
      Inc(vm);
      verem[vm].szint:=szint;
      verem[vm].kov:=kov;
      verem[vm].hossz:=hossz;
      verem[vm].szog:=szog;
    End
  End;
  Function UresVerem: Boolean;
  Begin
    UresVerem:=vm=0
  End;
```

```
  Procedure KochI1(szint,hossz:integer);
  Begin
    VeremInic;
    Verembe(szint,1,hossz,0);
    While not UresVerem do
    Begin
      Verembol(s,k,h,sz);
      If s>0 then
      Begin
        Case k of
          1: Begin   Verembe(s,k+1,h,+60);          Verembe(s-1,1,h Div 3,sz);     End;
          2: Begin   Verembe(s,k+1,h,-120);         Verembe(s-1,1,h Div 3,sz);     End;
          3: Begin   Verembe(s,k+1,h,+60);          Verembe(s-1,1,h Div 3,sz);     End;
          4: Begin   Verembe(s-1,1,h Div 3,sz);     End;
        End;
      End
        Else
      Begin
        TurnLeft(sz);    Forwd(h);
      End
    End;
  End;
```

```
  Procedure KochI2(szint,hossz:integer);
  Begin
    VeremInic;
    Verembe(szint, 1{ennek most nincs szerepe!!!}, hossz,0);
    While not UresVerem do
    Begin
      Verembol(s, k{ennek most nincs szerepe!!!},h, sz);
      If s>0 then
      Begin
        Verembe(s-1, 1,h Div 3, 60);      Verembe(s-1, 1,h Div 3, -120);
        Verembe(s-1, 1,h Div 3, 60);      Verembe(s-1, 1,h Div 3, sz {!!!});
      End
         Else
      Begin
        TurnLeft(sz);     Forwd(h);
      End;
    End;
  End;
```

```
Begin
   Inic;  Koch(3,300);   readln;
   Inic;  KochI1(4,300); readln;
   Inic;  KochI2(5,300); readln;
End.
```