

2.9. Az Object Pascal nyelv speciális lehetőségei

1. Lineáris lista [Lista](#)
2. Számrendszerváltás verem használatával [Stack](#)
3. Számrendszerváltás dinamikus tömbben tárolt veremmel [DinTomb](#)
4. Ansi és nullavégű sztringek megfordítása [Sfordito](#)
5. Forgalmi adatok tárolása dinamikus tömbben [Fokonyv](#)
6. Mandelbrot ábra rajzolása időméréssel [Idomero](#)
7. Polinom helyettesítési értékei [Polinom](#)
8. Pénzügyi számítások [JelenErtek](#)



A lista *TListaElem* rekordelemeket tartalmaz. A listaelemek tartalmaznak egy *TAdattipus* típusú egészet és egy mutatót a következő listaelemre.

```
type
  TAdattipus = integer;

  PListaElem = ^TListaElem;
  TListaElem = record           // A listaelem
    Info : TAdattipus;
    Link : PListaElem;
  end;
```

A programban a lista kezdetének azonosítására a *ListaFej* változót használjuk, amely tárolja a listaelemek számát is. Az aktuális listaelem a *pAkt* és a következő listaelem a *pKov*. Az *i*-t ciklusváltozónak használjuk.

```
var
  ListaFej : TListaElem;
  pAkt, pKov : PListaElem;
  i : integer;
```

Létrehozzuk az üres listafejet:

```
// listafej - a listában tárolt elemek számával
ListaFej.Info:=0;
ListaFej.Link:=nil;
```

Ciklusban felépítünk egy tízelemű listát.

```
// tízelemű lista felépítése
pAkt:=@ListaFej;           // Az aktuális mutató a listafejen
for i:=1 to 10 do
begin
  inc(ListaFej.info);      // A listaelemek számlálása
  new(pKov);               // Új listaelem létrehozása
  pAkt.Link:=pKov;         // Az aktuális listaelem a következőhöz láncolt
  pKov.Info:=i*i;          // A következő elem adata
  pKov.Link:=nil;          // A következő az utolsó elem
  pAkt:=pKov;              // Az aktuális listaelem az új elem
end;
```

A lista a fejtől indítva végigjárható:

```
// a lista bejárása, ameddig van listaelem
pAkt:=ListaFej.Link;
while Assigned(pAkt) do
begin
  writeln(pAkt.Info);
  pAkt:=pAkt.Link;        // Az aktuális listaelem az új elem
end;
```

Hasonlóan törölhető a lista:

```
// a lista elemeinek törlése, ameddig van listaelem
pAkt:=ListaFej.Link;
while Assigned(pAkt) do
begin
  pKov:=pAkt.Link;
  dispose(pAkt);           // Felszabadítás
  dec(ListaFej.info);
  pAkt:=pKov;              // Az aktuális listaelem az új elem
end;
ListaFej.Link:=nil;
```



Készítsünk 2..32 alapú számrendszerekbe való átváltáshoz használható programot! Az átváltás során keletkező osztási maradékokat tároljuk lineáris listával megvalósított veremben! (*Stack*)!

A feladatot úgy oldjuk meg, hogy a megadott (*szam*) számjegyet egészosztással elosztjuk az alapszámmal (*alap*), és a maradékot eltároljuk. A számot az osztás eredményével helyettesítve folytatjuk az alapszámmal való osztást és a maradék eltárolását, ameddig az osztás eredménye nem 0. Amikor a hányados 0, akkor az eltárolt maradékokat visszaolvasva megkapjuk a kiinduló számot az alapszám alapú számrendszerben.

A verem tárolást a [Lista](#)-hoz hasonlóan láncolt listával oldjuk meg, azonban a listaelemek ebben az esetben visszafelé láncoltak (mindegyik elem az előző elemre hivatkozik).

Szükségünk lesz a verembejegyzés típusára:

```
type emutato = ^elem;
    elem = record
        szamjegy : byte;
        elozo     : emutato;
    end;
```

Az alap és a szám tárolására az *alap* és *szam* változókat használjuk. Az *i* ciklusváltozó. A *verem* és a *seged* verem bejegyzés típusú pointerok. A számrendszerek számjegyeit a *jegyek* tömbben tároljuk.

```
var alap, szam      : int64;
    i : integer;
    verem, seged : emutato;
    jegyek : array[0..31] of char;
```

Első lépésben a számjegy karaktereket tároljuk (0,1,...,9,A,B...).

```
// a lehetséges számjegyeket tartalmazó tömb feltöltése:
for i:=0 to 9 do jegyek[i] := chr(48 + i);
for i:=10 to 31 do jegyek[i] := chr(55 + i);
```

Beolvassuk a számrendszer alapszámát és az átváltandó számot.

```
// a program adatainak beolvasása:
write('Kerem a szamrendszer alapjat:', #9);
readln(alap);
if not (alap in [2..32]) then
begin
    ShowMessage('Csak 2..32 közötti érték használható');
    halt;
end;
write('Az atvaltando szam:', #9#9);
readln(szam);
```

A beolvasott adatokkal, míg a 0-t el nem érjük, végezzük a maradékos osztást és a veremben való tárolást.

```
// a maradékok elhelyezése a veremben helyfoglalással:
verem:=nil;
while szam<>0 do
begin
    new(seged); // Új verembejegyzés
    seged^.szamjegy:= szam mod alap; // A maradékos osztás maradéka
    seged^.elozo := verem; // A verembejegyzések összeláncolása
    verem:= seged; // Beírás a verembe
    szam:= szam div alap; // Egézosztás
end;
```

A veremben tárolt elemek visszaolvasása és a verem felszabadítása a [Lista](#) programhoz hasonlóan történik.

```
// a szám kiírása a veremből történő visszaolvasással:
write('A(z) ', alap:2, ' számrendszerbeli alak:', #9);
seged:= verem;
while seged<>nil do                                // Végig a bejegyzéseken
begin
    write(jegyek[seged^.szamjegy]:1); // A számjegy kiírása
    seged:= seged^.elozo;
end;
writeln(#13#13);

// a veremterület felszabadítása:
seged:= verem;
while seged<>nil do                                // Végig a bejegyzéseken
begin
    verem:= seged^.elozo;                        // A láncszem kiszabadítása
    dispose(seged);                             // A bejegyzés felszabadítása
    seged:= verem;
end;
```



Írjunk 2.32 alapú számrendszerekbe való átváltáshoz használható alkalmazást! Az átváltás során keletkező osztási maradékokat tároljuk dinamikus tömbbel megvalósított veremben! (*DinTomb*)!

A feladat megoldásakor a [Stack](#) példa módszerét használjuk, egyetlen különbség, hogy veremnek a *verem* dinamikus tömböt használjuk. A verem hossza *vm*.

```
var
    verem : array of byte;
    vm    : integer;
```

Így sokkal egyszerűbb a verem méreteinek változtatása, a visszaolvasás és a felszabadítás is.

```
// a maradékok elhelyezése a veremben helyfoglalással:
SetLength(verem,0);
while szam<>0 do
begin
    vm := length(verem);
    SetLength(verem, vm+1);           // A verem méretének változtatása
    verem[vm] := szam mod alap;
    szam:= szam div alap;
end;

// a szám kiírása a veremből történő visszaolvasással
write('A(z) ', alap:2, ' számrendszerbeli alak:', #9);
for i:=length(verem)-1 downto 0 do
    write(jegyek[verem[i]]:1);
writeln(#13#13);

// a dinamikus tömb felszabadítása
SetLength(verem,0);
```



Készítsünk sztring karaktereit megfordító függvényeket! A **PChar** típusú paramétert fogadó függvény neve legyen **strrev**, míg a **string** típusúhoz használhatóé **stringrev**! (*Sfordito*)

Mindkét típus megfordítására eljárást írunk. Az eljárások hasonlóan működnek, a szöveg feléig egyszerűen kicseréljük az első félben és a második félben lévő karaktereket.

PChar esetén, mivel ez pointer, magában a sztringben dolgozunk. A szöveg hosszát az **strlen** függvény szolgáltatja.

```
// Nullavégű sztring megfordítása
procedure strrev(str : pchar);
var
  i,len : integer;
  ch    : char;
begin
  len:= strlen(str);
  for i:=0 to len div 2-1 do
    begin
      ch:= str[i];
      str[i]:= str[len-i-1];
      str[len-i-1]:= ch;
    end;
end;
```

Az *AnsiString* hosszát a **length** függvény szolgáltatja, és értékparaméterként kell átadni.

```
// AnsiString megfordítása
procedure stringrev(var str : string);
var
  i,len : integer;
  ch    : char;
begin
  len:= length(str);
  for i:=1 to len div 2 do
    begin
      ch:= str[i];
      str[i]:= str[len-i+1];
      str[len-i+1]:= ch;
    end;
end;
```

A főprogramban az **InputBox** függvénnyel az *s* változóba olvassuk be az adatot, és oda-vissza fordítjuk sztringként, illetve *PChar*-ként.

```
var
  s : string;
begin
  s:=inputbox('Adatbekérés','Kérem a megfordítandó szöveget',
    'Borland Delphi');
  stringrev(s);
  writeln(' A megfordított sztring      :'#9,s);

  strrev(pchar(s));
  writeln(' A visszafordított sztring :'#9,s);
  readln;
end.
```



Készítsünk alkalmazást, amely adott évben, adott darabszámú tranzakciót véletlenszerűen generál! Rendezzük dátum szerint az adatokat, majd jelenítsük meg azokat! (*Fokonyv*)

A véletlenszerűen létrehozott forgalmi adatokat *TAdat* típusú rekordban tároljuk.

```
type // A tranzakció adatait tároló adatstruktúra
  TAdat = record
    datum      : Tdate;
    forgalom   : currency;
  end;
```

Az éves adatok tárolására létrehozuk a *TEv* dinamikus tömb típust.

```
TEv = array of TAdat;
```

Az adatok feltöltését a *feltolt* eljárás végzi, az *aktev* paraméter a tranzakciók dinamikus tömbje, *trn* a tranzakciók száma.

```
// A tranzakciók létrehozása és dinamikus tömbben való tárolása
procedure feltolt(evsz : integer; var aktev : TEv; trn : integer);
const
  // A hónapok napjai
  napok:array[1..12] of byte = (31,0,31,30,31,30,31,31,30,31,30,31);
var
  i : integer;
  ev, ho, nap : word;
begin
  randomize;
  Setlength(aktev,trn);           // a tároló hosszának beállítása
  for i:=0 to trn-1 do
    begin
      ev:=evsz;
      if isLeapYear(ev) then      // szökőév-e
        napok[2]:= 29
      else
        napok[2]:=28;
      // Az adott évben belüli dátum véletlenszerű előállítás
      ho:=random(12)+1;
      nap:=random(napok[ho])+1;
      aktev[i].datum:=encodedate(ev,ho,nap); // A dátum TDateTime formában
      aktev[i].forgalom:=(2*random(2)-1)*random(maxint)/1e5;
    end;
  end;
```

Hasonló paraméterekkel rendelkezik az adatokat dátum szerint rendező *rendez* eljárás is. A rendezés során végignézzük az elem pozíciókat és minden egyes pozícióba a mögötte lévők közül a legkésőbbi kerül.

```
// A tranzakciók dátum szerinti sorbarendeze
procedure rendez(var aktev : TEv; trn : integer);
var
  i,j : integer;
  adat : TAdat;
begin
  for i:=0 to trn-2 do
    for j:=i+1 to trn-1 do
      if aktev[i].datum>aktev[j].datum then
        begin
          adat:=aktev[i];
          aktev[i]:=aktev[j];
          aktev[j]:=adat;
        end;
    end;
  end;
```

A kiíráshoz is eljárást használunk. A formátumok kialakításához az ezreket elválasztó jelet (*ThousandSeparator*) szóközre állítjuk és a *formatDateTime* és a *format* függvényeket használjuk.

```
// A tranzakciók megjelentése
procedure kiir(const aktev : TEv; trn : integer);
var
    i : integer;
begin
    ThousandSeparator:=#32;
    for i:=0 to trn-1 do
        begin
            write(formatDateTime('yyyy.mm.dd.',aktev[i].datum));
            writeln(format('%20.2m',[aktev[i].forgalom]));
        end;
    end;
```

Ezek után a főprogramban nem marad más dolgunk, mint a deklarált *ev* változóval a beolvasott évben (*evszam*), és a megadott tranzakciósámsra (*n*) aktivizáljuk a *feltolt*, a *rendez* és a *kiir* függvényeket.

```
// Főprogram
var
    ev : TEv;
    evszam, n :word;
begin
    write('A vizsgalt ev:');
    readln(evszam);
    write('A tranzakciok szama:');
    readln(n);
    feltolt(evszam, ev, n);
    rendez(ev, n);
    kiir(ev, n);
    readln;
end.
```




Írjunk programot, amely gombnyomásra megméri egy adott alprogram végrehajtási idejét, illetve többszöri gombnyomásra az időket átlagolja! Legyen a mért alprogram egy Mandelbrot ábrát megjelenítő eljárás (a szükséges programozási ismereteket az 5. fejezet tartalmazza)! (*Idomero*)

A Mandelbrot halmazt előállító eljárás:

```
// Mandelbrot halmazt megjelenítő eljárás
procedure Mandel(const frm : tform);
  var
    restep, imstep : extended;    // a valós és a képzetes rész lépéshossza
    repart, impart : extended;    // a valós és a képzetes rész
    x,y             : integer;     // az aktuális képernyő-koordináták
    maxx, maxy, maxc, cstep : integer; // grafikai jellemzők
    szín : integer;
  const
    restart = -2.267; reend=1.0;  // a valós rész értékhatárai
    imstart = -1.125;             // a képzetes rész kezdőértéke
    max : integer = 1600;         // a maximális iterációs mélység
    mag = 1;                     // magnifikációs tényező
    színek:array[0..15] of tcolor =(
                                   clblack, clNavy , clGreen ,clTeal, clMaroon,
                                   clPurple, clOlive, clSilver,
                                   clGray, clblue, clLime,
                                   clAqua, clRed, clFuchsia, clYellow, clWhite
                                   );

  // függvény az iteráció elvégzésére
  function iterate(re,im : extended): integer;
  var
    x,y,x2,y2 : extended;
    k: integer;
  begin
    x:=0; y:=0; x2:=y; y2:=0; k:=0;
    repeat
      y:=2*x*y+im;
      x:=x2-y2+re;
      x2:=sqr(x);
      y2:=sqr(y);
      inc(k);
    until (x2+y2 > 4) or (k>=max);
    iterate:=k;
  end;

  // függvény a kör/ciklois teszt elvégzésére
  function cycltest(zsx,re,im:extended):boolean;
  var zr,zs,imq:extended;
  begin
    imq := sqr(im);
    zr:=sqr(re) + imq;
    zs:=sqrt(zr-0.5*re+0.0625);
    if (16.0*zr*zs > 5*zs-4*re+1) and
      (sqr(re+1)+imq>0.0625) then cycltest:=true
      else cycltest:=false;
  end;

begin
  // indítási paraméterek beállítása
  maxx:=frm.clientwidth;
  maxy:=frm.clientheight;
  maxc:=15+1; cstep:=max div maxc;
  restep:=(reend-restart) /maxx;
  imstep:=restep;
```

```

// az iteráció indítása
y:=0;
impart:=imstart;
while y<=maxy do
begin
  repart := restart;
  x:=0;
  while x<=maxx do
  begin
    if cycltest(0.0625, repart, impart)
    then szin:=iterate(repart, impart)
    else szin:=max;
    frm.canvas.pixels[x,y]:=szinek[(maxc*cstep-szin) mod 16];
    inc(x, mag);
    repart:=repart+restep*mag;
  end; { while x }
  inc(y, mag);
  impart:=impart+imstep*mag;
end; {while y}
end;

```

Beépítettünk egy fél másodpercig várakozó tesztelő eljárást is.

```

// Átalgosan fél másodpercig várakozó eljárás
procedure Teszt;
begin
  Sleep(500);
end;

```

Az átlagok számításához bevezetünk két változót:

```

var
  atlagm : real=0;
  atlagc : real=0;

```

Építsünk a formra egy *Button1* és egy *Button2* gombot! Az előző indítsa a *Mandelbrot* iterációt, míg a második a *Teszt* várakozást. Mindkét esetben kiszámoljuk a szükséges időt és összevetjük az átlaggal.

```

// A rajzolás és időmérés gombnyomásra
procedure TForm1.Button1Click(Sender: TObject);
var
  start, vege :Tdatetime;
  h,p,mp,ms:word;
  ido : real;
begin
  Button1.enabled:=false;
  Button2.enabled:=false;
  refresh;
  caption:='Rajzol..';
  start:=time; // A kezdeti időpont a time változóból
  Mandel(form1);
  vege:=time; // A végzés időpontja a time változóból
  decodetime((vege-start),h,p,mp,ms); // az idő dekódolása
  ido:=(mp*100+ms)/100;
  if (atlagm=0) then atlagm:=ido
    else atlagm:=(atlagm+ido)/2;
  caption:='Mandelbrot idő: '+floattostr(ido)+' mp (Átlagidő: '+
    floattostr(atlagm)+ ' mp).';
  Button1.enabled:=true;
  Button2.enabled:=true;
end;

```

```

// Teszt időmérés gombnyomásra
procedure TForm1.Button2Click(Sender: TObject);
var
  start, vege :Tdatetime;
  h,p,mp,ms:word;
  ido : real;
begin
  Button1.enabled:=false;
  Button2.enabled:=false;
  caption:='Fut..';
  start:=time;      // A kezdeti időpont a time változóból
  Teszt;
  vege:=time;      // A végzés időpontja a time változóból
  decodetime((vege-start),h,p,mp,ms); // az idő dekódolása
  ido:=(mp*100+ms)/100;
  if (atlagc=0) then atlagc:=ido
                    else atlagc:=(atlagc+ido)/2;
  caption:='Fél másodperces tesztidő: '+floattostr(ido)+' mp (Átlagidő: '
                    +floattostr(atlagc)+ ' mp).';

  Button1.enabled:=true;
  Button2.enabled:=true;
end;

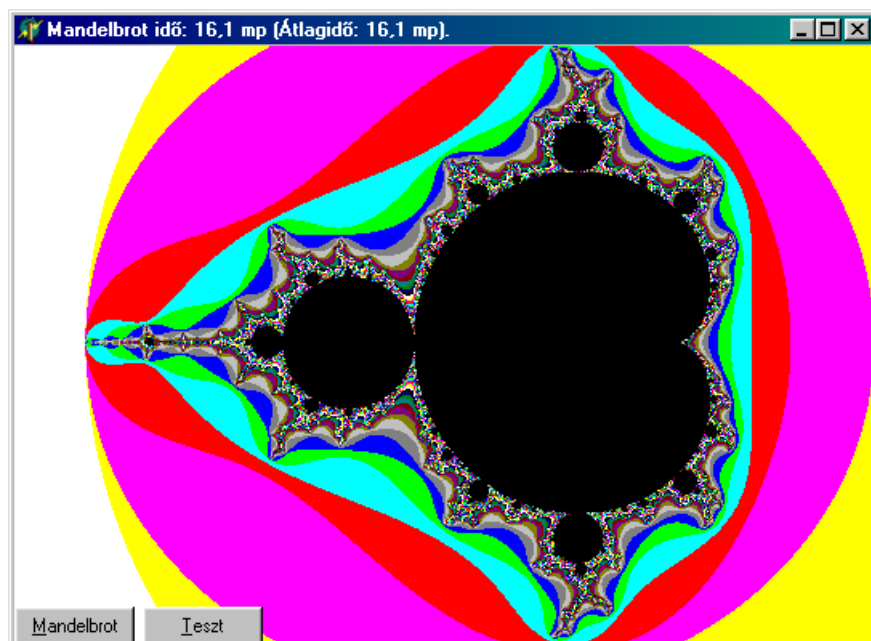
```

A form átméretezésnél töröljük az átlagokat

```

procedure TForm1.FormResize(Sender: TObject);
begin
  caption:='Időtesztek';
  atlagm:=0;
  atlagc:=0;
  refresh;
end;

```





A feladat megoldása során először beolvassuk a polinom fokszámát (*fox*). Az együtthatókat szintén beolvassuk és az *eh* dinamikus tömbben tároljuk. A független változó beolvasáskor az *x* változóba kerül. Minden beolvasás a *foxs* sztringbe történik, és csak aztán konvertáljuk számmá. Ha az együtthatónak beolvasott sztring nem konvertálható számmá, akkor az aktuális együttható 0 lesz. A program addig fut, míg számként értelmezhető független változót adunk meg. A polinom helyettesítési értékét a ***Poly*** függvénnyel számítjuk, amihez használnunk kell a ***Math*** modult.

A program listája:

```
program Polinom;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Math;
var
  fox,i,kod:integer;
  foxs      :string;
  x         :extended;
  eh        :array of Double;
begin
  Write('A polinom fokszama:');
  Readln(foxs);
  // a fokszámot stringbe olvassuk
  // csak akkor számolunk, ha értelmezhető
  val(foxs,fox,kod);
  if (kod=0) and (fox>=0) then
    begin
      SetLength(eh,fox+1);
      // beolvassuk az együtthatókat
      // ha nem értelmezhető az együttható 0
      for i:=0 to fox do
        begin
          Write('a[' ,i,']=');
          Readln(foxs);
          val(foxs,eh[i],kod)
        end;
      // Addig számoljuk a helyettesítési értéket,
      // míg értelmes a beolvasott független változó
      repeat
        Write('x=');
        readln(foxs);
        val(foxs,x,kod);
        if kod=0 then;
          Writeln('p(x)=',Poly(x,eh):16:4);
      until kod>0;
      finalize(eh);
    end;
end.
```



Készítsünk alkalmazást, melynek segítségével kiszámolhatjuk, mennyit kell befektetnünk ma, ha 5 év múlva 1 millió forintot szeretnénk kapni! A befektetés becsült átlagos jövedelmezősége 8,75% évente, az idő letele előtt a pénzből nem veszünk ki! (*JelenErtek*)

Az alkalmazásunk legyen egy egyszerű konzolalkalmazás! A számításhoz szükséges függvény a **Math** modul **PresentValue** függvénye, amely segítségével kiszámíthatjuk a befektetés jelenértékét a jövőbeli érték alapján. A függvény deklarációja a következő:

```
function PresentValue(Rate: Extended; NPeriods: Integer; Payment,
                      FutureValue: Extended; PaymentTime: TPaymentTime): Extended;
```

ahol a *Rate* paraméter a diszkontáláshoz használt rátát, illetve – a példánkban – a befektetés-jövedelmezőségi rátát, illetve a kamatlábat jelöli. Az *NPeriods* paraméterben az időszakok számát, a *Payment* paraméterben az időszakonkénti pozitív vagy negatív pénzáramlások értékét kell megadni – attól függően, hogy kivesszük-e a pénz, vagy hozzáadjuk az eredeti befektetés összegéhez. A példánkban a *Payment* paraméterben átadott érték 0 lesz, mivel időközben nem nyúlunk hozzá a pénzhez.

A **PresentValue** függvény *FutureValue* paraméterében a befektetés várható, illetve becsült jövőbeli értékét kell megadnunk, a *PaymentTime* paraméter értékével pedig jelezhetjük, hogy a pénzáramlások, illetve a kamatok kifizetése az időszakok elején (*ptStartOfPeriod*), vagy pedig végén (*ptEndOfPeriod*) történnek-e. Kamatfizetés (azaz mások által nekünk fizetendő pénz) esetén az érték egyértelműen a *ptEndOfPeriod* lesz.

Az alkalmazásunk fő modulja következőképpen néz ki:

```
program pvPr;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  pvu in 'pvu.pas';
begin
  Szamitas;
end.
```

A program által hívott egyetlen eljárás, a *Szamitas*, a *pvu* nevű modulban található, amelynek **implementation** szekciójához hozzá kell kapcsolni egyrészt az üzleti és pénzügyi függvények deklarációit tartalmazó **Math** modult, másrészt pedig a típuskonverziók (valós értékek karakterlánccá) elvégzéséhez szükséges **SysUtils** modult is:

```
unit pvu;
interface
  procedure Szamitas;

implementation
uses SysUtils, Math;

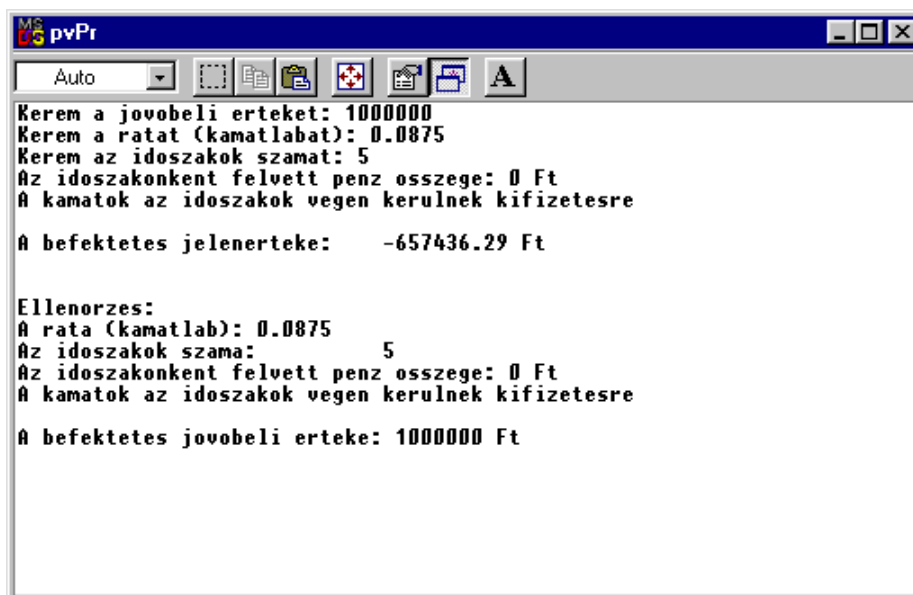
procedure Szamitas;
var
  jelenErtek, folyositando, jovobeliErtek, rata: Extended;
  evekSzama: Integer;
begin
  write('Kerem a jovobeli erteket: ');
  readln(jovobeliErtek);
  write('Kerem a ratat (kamatlabat): ');
  readln(rata);
  write('Kerem az idoszakok szamat: ');
  readln(evekSzama);
  folyositando:=0;
  writeln('Az idoszakonkent felvett penz osszege: '+FloatToStr(folyositando)+' Ft');
  writeln('A kamatok az idoszakok vegen kerulnek kifizetesre');
  writeln;
  jelenErtek:=PresentValue(rata, evekSzama, folyositando,
                           jovobeliErtek, ptEndOfPeriod);
  writeln('A befektetes jelenerteke: '+FloatToStr(jelenErtek)+' Ft');
  writeln;
```

```

readln;
writeln('Ellenorzes:');
writeln('A rata (kamatlab): '+FloatToStr(rata));
writeln('Az idoszakok szama: '+FloatToStr(evekSzama));
writeln('Az idoszakonkent felvett penz osszege: '+FloatToStr(folyositando)+' Ft');
writeln('A kamatok az idoszakok vegen kerulnek kifizetesre');
jovobeliErtek:=FutureValue(rata, evekSzama, folyositando, jelenErtek, ptEndOfPeriod);
writeln;
writeln('A befektetes jovobeli erteke: '+FloatToStr(jovobeliErtek)+' Ft');
readln;
end;
begin
end.

```

A *Szamitas* eljárás első részében elvégezzük a ma befektetendő összeg kiszámítását, a második részében pedig számításunk ellenőrzését, a *FutureValue* függvény segítségével. A *FutureValue* függvény működése és paraméterkészlete gyakorlatilag megegyezik a *PresentValue* függvényével, kivéve azt, hogy jelenérték helyett a befektetés jövőbeli értékét adja vissza, az ismert jelenérték alapján elvégezve a számítást.



```

MS pyPr
Auto
Kerem a jovobeli erteket: 1000000
Kerem a ratat (kamatlab): 0.0875
Kerem az idoszakok szamat: 5
Az idoszakonkent felvett penz osszege: 0 Ft
A kamatok az idoszakok vegen kerulnek kifizetesre
A befektetes jelenerteke: -657436.29 Ft
Ellenorzes:
A rata (kamatlab): 0.0875
Az idoszakok szama: 5
Az idoszakonkent felvett penz osszege: 0 Ft
A kamatok az idoszakok vegen kerulnek kifizetesre
A befektetes jovobeli erteke: 1000000 Ft

```

A fenti ábrán a program futásának eredménye látható. A jelenérték előtt álló mínuszjel azt jelzi, hogy a befektetés ma a mi kiadásunk.