

2.8 Modulok használata

1. Lineáris interpoláció: [Unit_pr1](#)
2. Műveletek komplex számokkal: [Unit_pr2](#)
3. Vektor koordináták összegének, átlagának számítása és maximális elemének megkeresése: [Unit_pr3](#)
4. Diákok tanulmányi átlagának számítása, és maximális kreditpontjának megkeresése: [Unit_pr4](#)
5. Raktárkészlet összértékének számítása, a legdrágább áru, minimál készlet keresése: [Unit_pr5](#)



Tervezzünk egy olyan programot, amely lineáris interpolációra alkalmas függvényt unit-ból használja.
(Unit_pr1)

A program bemutatja a lineáris interpoláció használatát. A *Modul* unit tartalmazza az alprogramokat és a szükséges *tomb* deklarációt. A *Unit_pr1* főprogram csak aktiválja azokat.

- Az *Olvas* eljárás megkérdezi a pontok számát (*n*), és két azonos hosszúságú vektort (*x,y* koordinátákat) olvas be a cím szerint átadott, paraméterként kapott *a* és *b tomb* típusú tömbbe (maximum húsz valós számpár). Megjegyezzük, hogy az áttekinthetőség kedvéért nem kezeltük a felhasználó esetleges hibáját. A program csak számokat olvas be.
- A *linint* függvénnyel pedig a lineáris interpolációt végezhetjük el. A függvény kikeresi a paraméterként kapott számpárok közül (*x, y* tömbök) azt a két *x*-koordinátát, melyek közrefogják a keresett független változót (*x1*). Ha nem sikerül, akkor a függvény igaz *hiba* visszatérési értékkel jelzi, hogy nem sikerült az interpoláció, egyébként a *hiba* érték hamis és a visszatérési érték az interpolált érték.

```
program Unit_pr1;
{$APPTYPE CONSOLE}

uses
  SysUtils,
  Modul in 'Modul.pas';

var
  xx,yy : tomb;
  db : integer;
  xakt, yakt : real;
  hiba : boolean;

begin
  Olvas(xx, yy, db);
  write('X koordinata: '); readln(xakt);
  yakt := linint(xx,yy,db,xakt,hiba);
  if not hiba then
    writeln('Interpolalt ertekek: ',xakt: 8:2,' helyen ',yakt:8:2)
  else
    writeln('Nincs interpolalt ertekek: hibas adat!');
  readln;
end.

unit Modul;

interface

type
  tomb = array[1..20] of real;
procedure Olvas(var a,b:tomb; var n: integer);
function linint(x,y:tomb; n:integer;x1:real; var hiba :boolean):real;

implementation

procedure Olvas(var a,b:tomb; var n: integer);
var
  i: integer;
begin
  writeln('Linearis interpolacio ');
  write('Alappontok szama: '); readln(n);
  writeln;
  writeln('X,Y interpolacios alappontok');
  for i:=1 to n do
    begin
      writeln;
      write(i:2,'. x koordinata: '); readln(a[i]);
      write(i:2,'. y koordinata: '); readln(b[i]);
    end;
  writeln;
end;

function linint(x,y:tomb; n:integer;x1:real; var hiba :boolean):real;
```

```

var
  i:integer;
  y1:real;
begin
  y1 := 0; hiba:=true;
  for i:=1 to n-1 do
    begin
      if (x[i] <= x1) and (x[i+1]>=x1) then
        begin
          if x[i+1] = x[i] then begin hiba := true; break; end;
          y1 := y[i]+(y[i+1]-y[i])/(x[i+1]-x[i])*(x1-x[i]);
          hiba := false;
        end;
      end;
    result:= y1;
  end;
end.

```



Készítsünk egy olyan programot, amely komplex számokkal végez műveleteket! A műveletek végrehajtásához használjunk eljárásokat és függvényeket is, szükséges eljárások és függvények legyenek külön modulban! (*Unit_pr2*)

A program bemutatja az összeadás, a kivonás és a szorzás műveleteket komplex számokkal. A műveletekhez szükséges alprogramokat a *Modul* könyvtár tartalmazza.

A *komplex* rekord típus hordozza a komplex számok valós és képzetes részét. A típus definícióját a *Modul* tartalmazza. Az egyes műveleteket az *osszead*, *kivon* és a *szorzat* alprogramok valósítják meg. Felhívjuk a figyelmet arra, hogy míg a *szorzat* függvény és visszatérési értékében szolgáltatja az eredményt, addig az *osszead* és a *kivon* cím szerinti paraméterében.

A számok beolvasására az *olvas*, kiírására a *kiir* eljárásokat használhatjuk.

A főprogram beolvas két komplex számot és kiírja a számok összegét, szorzatát és különbségét. Megjegyezzük, hogy az áttekinthetőség kedvéért nem kezeltük a felhasználó esetleges hibáját. A program csak számokat olvas be.

```
procedure olvas(var z: komplex);
  procedure osszead(x,y: komplex; var z: komplex);
  procedure kivon(x,y: komplex; var z: komplex);
  function szorzat(x,y: komplex):komplex;
  procedure kiir(z:komplex; s:string );
```

```
program Unit_pr2;
{$APPTYPE CONSOLE}
```

```
uses
  SysUtils,
  modul in 'modul.pas';
```

```
var
  a1,b1,ered1,ered2,szor: komplex;
begin
  writeln('Muveletek komplex szamokkal');
  writeln;
  olvas(a1);
  olvas(b1);
  osszead(a1,b1,ered1);
  kiir(ered1,'osszeadas');
  kivon(a1,b1,ered2);
  kiir(ered2,'kivonas');
  szor := szorzat(a1, b1);
  kiir(szor,'szorzas');
  readln;
end.
```

```
unit modul;
```

```
interface
```

```
type
  komplex = record
    re,im : real;
  end;
  procedure olvas(var z: komplex);
  procedure osszead(x,y: komplex; var z: komplex);
  procedure kivon(x,y: komplex; var z: komplex);
  function szorzat(x,y: komplex):komplex;
  procedure kiir(z:komplex; s:string );
```

```
implementation
```

```
  procedure olvas(var z: komplex);
  begin
    writeln;
```

```

    write('A komplex szam valos resze   : '); readln(z.re);
    write('A komplex szam kepzetes resze: '); readln(z.im);
end;

procedure osszead(x,y: komplex; var z: komplex);
begin
    z.re := x.re + y.re;
    z.im := x.im + y.im;
end;

procedure kivon(x,y: komplex; var z: komplex);
begin
    z.re := x.re - y.re;
    z.im := x.im - y.im;
end;

procedure kiir(z:komplex; s:string );
begin
    writeln;
    write('Muvelet: ',s:9,'   eredmenye:  ');
    writeln(z.re: 8:2,z.im:8:2);
end;

function szorzat(x,y:komplex):komplex;
begin
    result.re := x.re * y.re - x.im * y.im;
    result.im := x.im * y.re + x.re * y.im;
end;
end.

```



Írjunk egy olyan programot, amely kiszámítja a beolvasott vektor koordinátáinak összegét, átlagát és megkeresi a maximális elemét! A műveletekhez szükséges alprogramokat kerüljenek modulba! (*Unit_pr3*)

A program bemutatja a *Modul* unitban tárolt alprogramok használatát:

- a *VektorOlvas* eljárás feltölti x cím szerint hivatkozott a *tomb* típusú vektort n darab adattal,
- a *VektOsszeg* függvény kiszámítja és szolgáltatja a vektor elemeinek összegét,
- a *VektAtlag* eljárás a vektor átlagát adja vissza a kimenő paraméterében.
- a *MaxElem* függvény egy ciklus segítségével megkeresi és szolgáltatja a vektor legnagyobb elemét.

A főprogram csak meghívja a megfelelő alprogramokat. Megjegyezzük, hogy az áttekinthetőség kedvéért nem kezeltük a felhasználó esetleges hibáját. A program csak számokat olvas be.

```
program Unit_pr3;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Modul in 'Modul.pas';

var
  db: integer;
  a : tomb;
  Osszeg, Atlag : real;
begin
  VektOlvas(a,db);
  Osszeg := VektOsszeg(a,db);
  VektAtlag(a,db,atlag);
  writeln;
  writeln('A vektor elemeinek osszege: ',Osszeg:6:2);
  writeln('A vektor elemeinek atlaga : ',atlag:6:2);
  writeln('A vektor maximalis eleme : ',MaxElem(a,db):6:2);
  readln;
end.

unit Modul;
interface
type
  tomb = array[1..5] of real;

procedure VektOlvas(var x: tomb; var n: integer);
function VektOsszeg(x: tomb; n:integer): real;
procedure VektAtlag(x : tomb; n : integer; var atl: real);
function MaxElem(x: tomb; n:integer): real;

implementation

procedure VektOlvas(var x: tomb; var n: integer);
var
  i : integer;
begin
  write('A vektor elemeinek szama: '); readln(n);
  for i:=1 to n do
    begin
      write(i:2, '. elem: ');
      readln(x[i]);
    end;
  end;
end;

function VektOsszeg(x: tomb; n:integer): real;
var
  i : integer;
  s : real;
begin
  s := 0;
  for i:=1 to n do
    s:=s+x[i];
  VektOsszeg := s;
end;
```

```

procedure VektAtlag(x : tomb; n : integer; var atl: real);
begin
    atl:= VektOsszeg(x,n);
    atl := atl/n;
end;

function MaxElem(x: tomb; n:integer): real;
var
    i : integer;
begin
    result := x[1];
    for i:=2 to n do
        if result < x[i] then result := x[i];
    end;
end.

```



Tervezzünk egy olyan programot, amely a diákok tanulmányi átlagát számítja ki és megkeresi az évfolyamon elért maximális kreditpontot. (*Unit_pr4*)

A program bemutatja a *Modul* unitban tárolt alprogramok használatát:

- a *tanulo* rekord hivatott a hallgatók adatait tárolni,
- a *TankorAdatok* eljárás olvassa be a diákok adatait,
- a *KeresMaxKredit* függvény keresi meg az évfolyamon elért legnagyobb kreditpontot,
- az *ÉvfolyamAtlag* eljárás kiszámítja, és kimenő paraméterben szolgáltatja az évfolyam átlagát,
- a *NegyesAtlag* függvény a négyes és annál jobb átlaggal rendelkező diákok számát adja vissza.

A főprogram csak meghívja a megfelelő alprogramokat. Megjegyezzük, hogy az áttekinthetőség kedvéért nem kezeltük a felhasználó esetleges hibáját. A program a *tanulo* rekord számmezőibe csak számokat olvas be.

```
program Unit_pr4;
{$APPTYPE CONSOLE}

uses
  SysUtils,
  Modul in 'Modul.pas';
var
  tan_db, MaxKredit: integer;
  tanulok : tk;
  TanAtlag : real;

begin
  TankorAdatok(tanulok, tan_db);
  MaxKredit := KeresMaxKredit(tanulok, tan_db);
  EvfolyamAtlag(tanulok, tan_db, TanAtlag);
  writeln;
  writeln('Az evfolyam maximalis kreditpontja: ', MaxKredit);
  writeln('Az evfolyam atlaga : ', TanAtlag:6:2);
  writeln('A negyes atlag feletti tanulok szama: ',
    NegyesAtlag(tanulok, tan_db));
  readln;
end.

unit Modul;

interface

type
  tanulo = record
    neve : string[40];
    tankor : integer;
    kredit: integer;
    atlag : real;
  end;
  tk = array[1..20] of tanulo;

procedure TankorAdatok(var x: tk; var n: integer);
function KeresMaxKredit(x: tk; n: integer): integer;
procedure EvfolyamAtlag(x : tk; n : integer; var EvfAtl: real);
function NegyesAtlag(x: tk; n: integer): integer;
```



```

implementation

procedure TankorAdatok(var x: tk; var n: integer);
var
    i : integer;
begin
    write('A tanulok szama: '); readln(n);
    writeln;
    for i:=1 to n do
    begin
        writeln(i:2, '. tanulo: ');
        write('Neve      : '); readln(x[i].neve);
        write('Tankor   : '); readln(x[i].tankor);
        write('Kredit   : '); readln(x[i].kredit);
        write('Atlag    : '); readln(x[i].atlag);
        writeln;
    end;
end;

function KeresMaxKredit(x: tk; n:integer): integer;
var
    i : integer;
    kr : integer;
begin
    kr := x[1].kredit;
    for i:=2 to n do
        if x[i].kredit> kr then kr := x[i].kredit;
    KeresMaxKredit := kr;
end;

procedure EvfolyamAtlag(x : tk; n : integer; var EvfAtl: real);
var
    i : integer;
begin
    EvfAtl:= 0;
    for i:=1 to n do
        EvfAtl:= EvfAtl + x[i].atlag;
    EvfAtl := EvfAtl/n;
end;

function NegyesAtlag(x: tk; n: integer):integer;
var
    i: integer;
begin
    result := 0;
    for i:=1 to n do
        if x[i].atlag >= 4 then result := result+1;
    end;
end.

```



Készítsünk egy olyan programot, amely kiszámítja a raktárkészlet összértékét, megkeresi a legdrágább és a minimális készlettel rendelkező árut! (*Unit_pr5*)

A program bemutatja a *Modul* unitban tárolt alprogramok használatát:

- az áruk adatait az *aru* típusú rekord tartalmazza,
- a *RaktarFeltolt* eljárással olvassuk be a raktár adatait,
- a *LegdragabbAru* függvény megkeresi a raktárban lévő áruk közül a legdrágábbat,
- a *MinimalKeszlet* függvény pedig annak az árunak az indexét adja vissza, melynek legkisebb a súlya,

```
program Unit_pr5;
{$APPTYPE CONSOLE}

uses
  SysUtils,
  Modul in 'Modul.pas';

var
  r      : raktar;
  r_db ,index: integer;
  OsszErtek: real;
  MaxAr: real;

begin
  RaktartFeltolt(r,r_db);
  OsszErtek := RaktarErteke(r,r_db);
  writeln('A raktar osszerteke : ',OsszErtek:8:2);
  MaxAr := LegdragabbAru(r,r_db);
  writeln('A legdragabb aru      : ',MaxAr:8:2);
  index := MinimalKeszlet(r,r_db);
  writeln('Minimalis keszlet van: ');
  writeln('                azonosito: ',r[index].azonosito);
  writeln('                sulya:      ',r[index].sulya:8:2);
  readln;
end.

unit Modul;
interface

const maxaru = 30;
type
  aru = record
    azonosito : string;
    ara       : real;
    sulya     : real;
  end;
  raktar = array[1..maxaru] of aru;

  procedure RaktartFeltolt(var x:raktar; var n:integer);
  function  RaktarErteke(x:raktar; n: integer):real;
  function  LegdragabbAru(x:raktar; n:integer):real;
  function  MinimalKeszlet(x:raktar; n: integer):integer;
  function  MinimalKeszletDB(x:raktar; n: integer; j: integer):integer;

implementation
  procedure RaktartFeltolt(var x:raktar; var n:integer);
  var
    i : integer;
  begin
    write('Arufajtak szama: '); readln(n);
    for i:=1 to n do
      begin
        writeln;
        write('Aru azonosito      : '); readln(x[i].azonosito);
        write('Aru fajlagos ara      : '); readln(x[i].ara);
        write('Aru sulya                : '); readln(x[i].sulya);
      end;
      writeln;
    end;
  end;
```

```

function RaktarErteke(x:raktar; n: integer):real;
var
  i : integer;
begin
  result := 0;
  for i:=1 to n do
    result := result + x[i].ara*x[i].sulya;
  end;

function LegdragabbAru(x:raktar; n:integer):real;
var
  i: integer;
begin
  result := x[1].ara;
  for i:=2 to n do
    if result < x[i].ara then result := x[i].ara;
  end;

function MinimalKeszlet(x:raktar; n: integer):integer;
var
  i : integer;
  suly: real;
begin
  suly := x[1].sulya;
  result := 1;
  for i:=2 to n do
    if suly > x[i].sulya then
      begin
        suly := x[i].sulya;
        result := i;
      end;
  end;

end.

```