Rekurzió

Tartalomjegyzék

Készítette: Gál Tamás

Közzétéve: http://informatika.gtportal.eu/

Rekurzió

Rekurzióról beszélünk, ha egy eljárás, függvény vagy metódus közvetve vagy közvetlenül önmagát hívja.

Rekurzió célja egy összetett feladat visszavezetése egyszerűbb feladatok ismételt megoldására. Egy alprogram működése változatlan, de mindig saját magát hívja, de egyre egyszerűbb argumentumokkal.

A ciklusokhoz hasonló módon rekurzió esetén is gondoskodni kell arról, hogy véges lépésszám után leálljon az alprogram ismételt meghívása, különben a program a végtelen ciklushoz hasonló módon viselkedik.

A következő példakód hasonlóképp működik mint egy növekményes ciklus, amelynek ciklusváltozója 10-től indulva 1-ig csökken (0-nál kilép).

Először meghívva az eljárást i=10. Ezt képernyőre írja, majd 1-el kisebb értékkel hívja saját magát. Mindaddig ismétli a folyamatot (kiírás, saját maga meghívása) amíg i>0. 0-nál leáll.

Mondatszerű leírás:

```
Függvény ismétel(i:egész típus):
  Ha i>0 akkor
    KI: i
    ismétel(i-1)
  Elágazás vége
Függvény vége.
```

Pascal kód:

```
program Rekurzio;

procedure ismetel(i:integer);
begin
  if i>0 then begin
    writeln(i);
    ismetel(i-1);
  end;
end;

begin
 ismetel(10);
end.
```

Java kód:

```
...
public static void main(String[] args) {
   ismetel(10);
}

static void ismetel(int i){
   if (i>0) {
      System.out.println(i);
      ismetel(i-1);
   }
}
...
```

Kérdések, feladatok

- Módosítsd az „`ismetel`” nevű eljárást, úgy

Hatványozás rekurzióval

Specifikáció:

$$x^n = \begin{cases} x * x^{n-1} & \text{ha } n > 0 \\ 1 & \text{ha } n = 0 \end{cases}$$

Mondatszerű leírás:

```
Függvény Hatvány(x: valós, n: egész): valós
  Ha n > 0 akkor
     Hatvány := x * Hatvány(x, n-1)
  különben
     Hatvány := 1
  Elágazás vége
Függvény vége.
```

Pascal kód:

```
program Hatvanyozas;

function hatvany(x:Double; n:integer): Double;
begin
  if n>0 then
    hatvany := x * hatvany(x, n-1)
  else
    hatvany := 1;
  writeln('x:',x:3:0,'  n:',n:2,'  hatvány:',hatvany:4:0)
end;

begin
 writeln(hatvany(2,10):4:0);
end.
```

Java kód:

```
...
public static void main(String[] args) {
   System.out.println(hatvany(2,10));
}

static float hatvany(float x, int n){
   if (n>0) {
      return x * hatvany(x, n-1);
   } else {
      return 1;
   }
}
...
```

Faktoriális számítás rekurzióval

Az $n \in N$ faktoriális (jelölés: n!) alatt az első n darab pozitív természetes szám szorzatát értjük, n=0 esetén pedig 1-et.

Specifikáció:

$$n! = \begin{cases} n * (n - 1)! & \text{ha } n > 0 \\ 1 & \text{ha } n = 0 \end{cases}$$

Mondatszerű leírás:

```
Függvény Faktoriális(n: egész): egész
  Ha n > 0 akkor
     Faktoriális := n * Faktoriális(n – 1)
  különben
     Faktoriális := 1
  Elágazás vége
Függvény vége.
```

Működés n=4 esetén:

Faktoriális(4) =
4 * Faktoriális(3) = (verembe: 4, és visszatérési cím)
3 * Faktoriális(2) = (verembe: 3, és visszatérési cím)
2 * Faktoriális(1) = (verembe: 2, és visszatérési cím)
1 * Faktoriális(0) (verembe: 1, és visszatérési cím)
Faktoriális(0)=1
1 * Faktoriális(0) (1*1= 1) (veremből: 1, és visszatérési cím)
2 * Faktoriális(1) = (2*1= 2) (veremből: 2, és visszatérési cím)
3 * Faktoriális(2) = (3*2= 6) (veremből: 3, és visszatérési cím)
4 * Faktoriális(3) = (4*6=24) (veremből: 4, és visszatérési cím)

Faktoriális(4) = 24

[forrás: Petrik tananyagtár: Rekurzió]

A rekurzió folyamata:

- a függvény a faktoriális számítását visszavezeti 1-gyel kisebb egész szám faktoriálisának a kiszámítására, amíg lehet.
- a számítás elvégzése közben újra és újra meghívja magát, miközben a számítás folytatásához szükséges értékeket és a visszatérési címeket verembe teszi.
- a folyamat legalsó szintjét a Faktoriális(0) jelenti, amelynek az értéke definíció szerint 1, itt ér véget a rekurzív hívások sorozata.
- ezután folyamatos visszahelyettesítések következnek, a folyamat minden szintjén, és a legfelső szinten meghatározásra kerül n! értéke.

[forrás: Petrik tananyagtár: Rekurzió]

Pascal kód:

```
program Faktorialis1;

function Faktorialis(n:integer): integer;
begin
  if n>0 then
    Faktorialis := n * Faktorialis(n - 1)
  else
    Faktorialis := 1;
  writeln(Faktorialis);
end;

begin
 writeln(Faktorialis(4));
end.
```

Java kód:

```
...
public static void main(String[] args) {
   System.out.println(Faktorialis(5));
}

static int Faktorialis(int n){
   if (n>0) {
      return n * Faktorialis(n-1);
   } else {
      return 1;
   }
}
...
```

Kérdések, feladatok

- Módosítsd az „`Faktorialis`” nevű alprogramot úgy, hogy a képernyőn visszatérés előtt jelenítse meg visszatérési értékét!
- A `Faktorialis` nevű alprogram hányszor hívja meg saját magát, ha n paraméterének értéke kezdetben 5?

Fibonacci sorozat

A Fibonacci-sorozat első két eleme 0 és 1, a további elemeket az előző kettő összegeként kapjuk.

Specifikáció:

$$Fibonacci_n = \begin{cases} Fibonacci_{n-1} + Fibonacci_{n-2} & \text{ha } n > 1 \\ n & \text{ha } n=0 \\ n & \text{ha } n=1 \end{cases}$$

Mondatszerű leírás:

```
Függvény Fibonacci(n: egész): egész
  Ha n > 1 akkor
    Fibonaccin := Fibonacci(n - 1) + Fibomacchi(n - 2)
  különben
    Fbonacchi := n
  Elágazás vége
Függvény vége
```

Pascal kód:

```
program Fibonacci1;

function Fibonacci(n:integer): integer;
begin
  if n>1 then
    Fibonacci := Fibonacci(n - 1) + Fibonacci(n - 2)
  else
    Fibonacci := n;
end;

begin
 writeln(Fibonacci(4));
end.
```

Java kód:

```
...
public static void main(String[] args) {
   System.out.println(Fibonacci(5));
}

static int Fibonacci(int n){
   if (n>1) {
      return Fibonacci(n - 1) + Fibonacci(n - 2);
   } else {
      return n;
   }
}
...
```

Kérdések, feladatok

- Módosítsd az „`Fibonacci`” nevű alprogramot úgy, hogy a képernyőn visszatérés előtt jelenítse meg visszatérési értékét!
- A `Faktorialis` nevű alprogram hányszor hívja meg saját magát, ha n paraméterének értéke kezdetben 6?

QuickSort (gyorsrendezés)

A leghatékonyabb rendezési algoritmus.

A tömb számtanilag középső elemét kiemelve, balról megkeressük az első olyan elemet, amely nagyobb vagy egyenlő a középső elemmel, majd jobbról az első kisebb vagy egyenlő elemet. Ha találunk 2 megfelelő elemet, akkor kicseréljük. A ciklus végére a kiemelt elemtől balra csak nála kisebb, jobbra pedig nagyobb elemek állnak.

Rekurzív módon megismételjük a folyamatot a bal és a jobb oldali tartományra is. Egy sorozat rendezését rendre két kisebb részének rendezésére vezetjük vissza.

Specifikáció:
a:tömb(n) – a rendezendő tömb
Bal, Jobb: egészek – a szétválogatásra kerülő rész eleje és vége, bemenő paraméterek
k: egész – az elválasztó elem pozíciója, kimenő paraméter

Mondatszerű leírás:

```
Eljaras Quicksort(Bal, Jobb)
i := Bal
j := Jobb
Kozepsokulcs := A[(i+j)/2]
Ciklus amíg i <= j
  Ciklus amíg A[i].K< Kozepsokulcs
    i := i + 1
  Ciklus vége
  Ciklus amíg A[j].K > Kozepsokulcs
    j := j - 1
  Ciklus vége
  Ha i<j akkor
    Csere( A[i] , A[j] )
    i:=i+1
    j:=j-1
  Elágazás vége
Ciklus vége
Ha Bal < j    akkor Quicksort(Bal, j-1)
Ha i   < Jobb akkor Quicksort(i+1, Jobb)
Eljárás vége
```

Pascal kód:

```
Program  QuicksortPrg;
const n = 5;
var
  a: array[1..n] of integer = (135,50,-42,44,21);
  i, k: integer;
procedure quicksort(Bal,Jobb:integer);
var i,j,k,Pivot,x:integer;
begin
 i:=Bal;
 j:=Jobb;
 Pivot:=a[(i+j) div 2];

 while i<=j do
  begin
   while a[i]<Pivot do inc(i);
   while a[j]>Pivot do dec(j);
   if i<=j then begin
                x:=a[i];
                a[i]:=a[j];
                a[j]:=x;
                inc(i);
                dec(j);
   end;
 end;
 if Bal<j then quicksort(Bal,j);
 if i<Jobb then quicksort(i,Jobb);
end;

begin
   quicksort(1,n);
   for i:=1 to n do writeln(i,'. elem:',a[i]);
end.
```

Java kód:

```
...
 public static void main(String[] args) {
   int[] a = {135,50,-42,44,21};
   quickSort(a,0,a.length-1);
   System.out.println(a);
        for(int i:a){
            System.out.print(i);
            System.out.print(" ");
        }
 }
 static void quickSort(int[] a, int lowerIndex, int higherIndex) {
  int i = lowerIndex;
  int j = higherIndex;
  int pivot = a[lowerIndex+(higherIndex-lowerIndex)/2];
  while (i <= j) {
    while (a[i] < pivot) {i++;}
    while (a[j] > pivot) {j--;}
    if (i <= j) {
      int temp = a[i];
      a[i] = a[j];
      a[j] = temp;
      i++;
      j--;
    }
  }
  if (lowerIndex < j)     quickSort(a, lowerIndex, j);
  if (i < higherIndex)    quickSort(a, i, higherIndex);
}
...
```

Ajánlott: https://users.itk.ppke.hu/~kami/Oktatasi_Anyagok/Szakiranyu%20Tovabbkepzes/06/05_01_Rendez%E9sek_2.pdf

Kérdések, feladatok

- Módosítsd az „`quicksort`” nevű alprogramot úgy, hogy csökkenő sorrendbe rendezze a tömb elemeit!
- Módosítsd az „`quicksort`” nevű alprogramot úgy, hogy minden meghívásakor megjelenítse paraméterként kapott tömbindexeket, a kiemelt tömbelem értékét, és a tömbelemek aktuális sorrendjét! Legalább 2 számsor esetén írd le füzetedbe az megjelenített adatokat áttekinthető formában és jelöld a változásokat.

Iteratív bináris keresés

Rendezett tömbben nagyon gyors keresés tesz lehetővé.
Megvizsgálja a tömb középső elemét.
Három eset lehetséges:

- megtalálta a keresett elemet >> visszaadja indexét
- kisebb a kereset elemnél >> a tömbnek már csak az aktuális elemet követő részét vizsgálja
- nagyobb a kereset elemnél >> a tömbnek már csak az aktuális elemet megelőző részét vizsgálja

Specifikáció:
a: tömb(n) – a rendezett tömb
Elso,Utolso: egészek – a vizsgált kerülő rész eleje és vége, bemenő paraméterek
keresett: egész – a keresett érték, bemenő paraméter
Van: logikai – cím szerint átadott paraméter
Index: egész – cím szerint átadott paraméter – a kereset elem indexe

---

Mondatszerű leírás:

```
ciklus amíg  első <= utolsó  és  van = hamis
    Középső :=  (Első + Utolsó)  Div 2
    Ha keresett = t[középső] akkor
      van := igaz
      index := középső
    else
        ha Keresett < t[középső] akkor
            utolsó := Középső - 1
        Ellenben
            Első := Középső + 1
        Ha vége
ciklus vége
```

Pascal kód:

```
Program  BinarisKeresesPRG;
const n = 5;
type
  TmbTip =  array[1..n] of integer;
var
  a: array[1..n] of integer = (-42,21,44,50,135);
  i, k, SORSZ: integer;
  OK : boolean;

Procedure BinarisKereses(n:Integer; keresett:integer; Var a:TmbTip;
                         Var Van:Boolean; Var Index:Integer);
var Elso,Utolso,Kozep:Integer;
begin
  Elso:=1; Utolso:=n;  Van:=false;
   while ((Elso<=Utolso) AND (Van=false)) do begin
     kozep:= (Elso+Utolso ) div 2;

     If keresett=A[kozep] then begin
       Van:=true;
       Index:=kozep;
     end else begin
       If keresett<A[kozep] then Utolso:=kozep-1
                            else Elso  :=kozep+1;
     end;
  end;
end;
```

```
begin
   BinarisKereses(n,135,a,OK,SORSZ);
   if (OK) then writeln('Sorszám:',SORSZ);
   for i:=1 to n do writeln(i,'. elem:',a[i]);
end.
```

Java kód:

```
...
public static void main(String[] args) {
  int[] a = {-42,21,44,50,135};
  int index = BinarisKereses(a,a.length-1,44);
  System.out.println(a);
  for(int i:a){
    System.out.print(i);
    System.out.print(" ");
  }
  System.out.println(" ");
  System.out.print(index);
}

static int BinarisKereses(int[] a, int n, int keresett)
{
  int bal = 0;
  int jobb = n - 1;
  for(int kozep = (bal+jobb)/2; bal <= jobb; kozep = (bal+jobb)/2)
  {
    if (a[kozep] == keresett)
    {
      return kozep;
    }
    else
    {
      if (keresett < a[kozep])
      {
        jobb = kozep - 1;
      }
      else
      {
        bal = kozep + 1;
      }
    }
  }
  return -1;
}
...
```

Rekurzív bináris keresés

Ha a középső elem kisebb vagy nagyobb a keresettnél, akkor rekurzív hívásokkal vizsgáljuk a tömb mögötte vagy előtte található elemeit.

Specifikáció:
a: tömb(n) – a rendezett tömb
bal, jobb: egészek – a vizsgált kerülő rész eleje és vége, bemenő paraméterek
keresett: egész – a keresett érték, bemenő paraméter
visszatérési érték – a keresett elem pozíciója, ha nem található, akkor -1

Mondatszerű leírás:

```
Függvény RekurzivBinKer(a: tömb; keresett, bal, jobb: egész): egész
  Ha jobb < bal akkor
     RekurzivBinKer := -1
  különben
    kozep := (bal + jobb) / 2;
    Ha (a[kozep] = keresett) akkor
      RekurzivBinKer := kozep
    különben
      Ha (keresett < a[kozep]) akkor
        RekurzivBinKer :=  RekurzivBinKer(a, keresett, bal, kozep – 1)
      különben
        RekurzivBinKer :=  RekurzivBinKer(a, keresett, kozep + 1, jobb);
      Elágazás vége
    Elágazás vége
  Elágazás vége
Függvény vége
```

Pascal kód:

```
Program  BinarisKeresesPRG;
const n = 5;
type
  TmbTip =  array[1..n] of integer;
var
  a: array[1..n] of integer = (-42,21,44,50,135);
  i, k, Index: integer;
  OK : boolean;

function RekurzivBinKer(Var a:TmbTip; keresett, bal, jobb:integer): integer;
var kozep:integer;
begin
  if (jobb < bal) then RekurzivBinKer := -1 else begin
    kozep := (bal + jobb) div 2;
    if (a[kozep] = keresett) then
      RekurzivBinKer := kozep
   else
     if (keresett < a[kozep]) then
       RekurzivBinKer :=  RekurzivBinKer(a, keresett, bal, kozep - 1)
     else
       RekurzivBinKer :=  RekurzivBinKer(a, keresett, kozep + 1, jobb);
  end;
end;

begin
   Index := RekurzivBinKer(a,135,0,n);
   if Index>-1 then writeln('Sorszám:',Index) else writeln('Nincs');
   for i:=1 to n do writeln(i,'. elem:',a[i]);
end.
```

Java kód:

```
...
public static void main(String[] args) {
  int[] a = {-42,21,44,50,135};
  int index = RekurzivBinKer(a,50,0,a.length-1);
  System.out.println(a);
     for(int i:a){
        System.out.print(i);
        System.out.print(" ");
     }
  System.out.println(" ");
  System.out.print(index);
}

static int RekurzivBinKer(int[] a, int keresett, int bal, int jobb)
{
  if (jobb < bal) return -1;
  int kozep = (bal + jobb) / 2;
  if (a[kozep] == keresett)
  {
    return kozep;
  }
  else
  {
    if (keresett < a[kozep])
    {
      return RekurzivBinKer(a, keresett, bal, kozep - 1);
    }
    else
    {
      return RekurzivBinKer(a, keresett, kozep + 1, jobb);
    }
  }
}
...
```

Hanoi-tornyai

A játék szabályai szerint az első rúdról az utolsóra kell átrakni a korongokat úgy, hogy minden lépésben egy korongot lehet áttenni, nagyobb korong nem tehető kisebb korongra, és ehhez összesen három rúd áll rendelkezésre.

Specifikáció:

```
n: korong(ok) száma
Forras: melyik pálcáról kell, hogy átrakjuk a korongokat
Cel: melyik pálcára kell, hogy átrakjuk a korongokat
Seged: melyik pálca segítségével rakjuk át a korongokat
```

Mondatszerű leírás:

```
Eljárás Hanoi(n, Forras, Cel, Seged)
  Ha n > 0 akkor
     Hanoi(n-1, Forras, Seged, Cel)
  KI: n ”korongot”  Forras > Cel
  Ha n > 0 akkor
     Hanoi(n-1, Seged, Cel, Forras)
Eljárás vége
```

```
program HanoiPrg;

Procedure Hanoi(n,Forras,Seged,Cel:integer);
 Begin
  if n>0 then begin
     Hanoi(n-1,Forras,Cel,Seged);
     Writeln(n,'. korongot ',Forras,'. rúdról ',Cel,'. rúdra!');
     if (n > 1) then Hanoi(n-1,Seged,Cel,Forras);
  end;
 End;

begin
  Hanoi(3,1,2,3);
end.
```

```
program HanoiPrg;

procedure Hanoi(i, n: integer; Forras, Cel, Seged: char);
begin
  i:=i+1;
  writeln('i:',i,' n:',n,' Forras:', Forras, ' Cel:', Cel, ' Seged:',Seged);
  if n > 0 then begin                { ha van korong }
    Hanoi(i,n-1, Forras, Seged, Cel);   { n-1 -et a 'tanitvany' átrak a Seged-re }
    writeln(n:2,'. ',Forras,'->',Cel);{ helyere rakjuk az felsőt }
    Hanoi(i,n-1, Seged, Cel, forras);   { a 'tanitvany' a Seged-ről a helyére tesz }
  end
end;

begin
  Hanoi(0,3,'A','B','C');
end.
```

Hanoi-tornyai

Hanoi-tornyai játék rúdjaira mindig csak felülről helyezhetünk új korongokat, és csak a legfelsőt tudjuk leemelni. Ez a megfelel a verem memóriák működésének. A következő program minden rúdhoz egy rekordot rendel, amiben külön mező tárolja a veremmutató értékét, a másik mezőben található tömb pedig a benne elhelyezett korongok sorszámát.

A Push eljárás az adott verem tetejére helyezi a paraméterben kapott sorszámot, és növeli a veremmutatót, ha még nem értük el a verem tetejét.

A Pop függvény az adott verem tetejéről leveszi az ott található sorszámot, és csökkenti a veremmutatót, feltéve ha verem nem üres.

```
program HanoiVerem;
const
  db = 4;
type
VeremT=Record
  Mut:integer;
  Elemek:array[1..db] of integer;
end;
var
  AVerem, BVerem, CVerem: VeremT;

procedure InitVermek;
var
  i : integer;
begin
   Averem.Mut := db;
   Bverem.Mut := 0;
   Cverem.Mut := 0;
   for i:=1 to db do begin
     Averem.Elemek[i] := i;
     Bverem.Elemek[i] := 0;
     Cverem.Elemek[i] := 0;
   end;
end;

procedure PushA (n:integer);
begin
   if Averem.Mut<db+1 then begin
      Averem.Mut :=  Averem.Mut + 1;
      Averem.Elemek[Averem.Mut] := n;
   end;
end;

function PopA: integer;
begin
   if Averem.Mut>0 then begin
      PopA  := Averem.Elemek[Averem.Mut];
      Averem.Elemek[Averem.Mut] := 0;
      Averem.Mut :=  Averem.Mut - 1;
   end else PopA := 0;
end;
```

```
procedure KiirAverem;
var
  i : integer;
begin
  for i:= 1 to db do write(Averem.Elemek[i]);
  writeln();
end;

procedure PushB (n:integer);
begin
   if Bverem.Mut<db+1 then begin
      Bverem.Mut :=  Bverem.Mut + 1;
      Bverem.Elemek[Bverem.Mut] := n;
   end;
end;

function PopB: integer;
begin
   if Bverem.Mut>0 then begin
      PopB  := Bverem.Elemek[Bverem.Mut];
      Bverem.Elemek[Bverem.Mut] := 0;
      Bverem.Mut :=  Bverem.Mut - 1;
   end else PopB := 0;
end;

procedure KiirBverem;
var
  i : integer;
begin
  for i:= 1 to db do write(Bverem.Elemek[i]);
  writeln();
end;

procedure PushC (n:integer);
begin
   if Cverem.Mut<db+1 then begin
      Cverem.Mut :=  Cverem.Mut + 1;
      Cverem.Elemek[Cverem.Mut] := n;
   end;
end;

function PopC: integer;
begin
   if Cverem.Mut>0 then begin
      PopC  := Cverem.Elemek[Cverem.Mut];
      Cverem.Elemek[Cverem.Mut] := 0;
      Cverem.Mut :=  Cverem.Mut - 1;
   end else PopC := 0;
end;

procedure KiirCverem;
var
  i : integer;
begin
  for i:= 1 to db do write(Cverem.Elemek[i]);
  writeln();
end;

procedure Atrak(Forras, Cel:char);
var
  n:integer;
begin
  Case Forras of
    'A' : n := PopA;
    'B' : n := PopB;
```

```
    'C' : n := PopC;
  end;
  Case Cel of
    'A' : PushA(n);
    'B' : PushB(n);
    'C' : PushC(n);
  end;
  writeln('Átrak ', Forras, '(',n,') >', Cel);
  KiirAverem;
  KiirBverem;
  KiirCverem;
  writeln();
end;

procedure Hanoi(i, n: integer; Forras, Cel, Seged: char);
begin
  i:=i+1;
  {writeln('i:',i,' n:',n,' Forras:', Forras, ' Cel:', Cel, ' Seged:',Seged);}
  if n > 0 then begin                    { ha van korong }
    Hanoi(i,n-1, Forras, Seged, Cel);
    Atrak(Forras, Cel);
    Hanoi(i,n-1, Seged, Cel, forras);
  end
end;

begin
  InitVermek;
  KiirAverem;
  Hanoi(0,db,'A','B','C','D');
  KiirAverem;
  KiirBverem;
  KiirCverem;
end.
```

Kérdések, feladatok

- Próbáld ki a „HanoiVerem” nevű programot ! Ellenőrizd működését 3,4 és 5 korong esetén!
- Készítsd el a verem memóriákat kezelő alprogramok mondatszerű leírásait!