

Tartalomjegyzék

Algoritmusok - pszeudókód	1–28
Abszolút érték	1
Hányados ismételt kivonással	1
Legnagyobb közös osztó	1
Páros számok szűrése	1
Palindrom számok	2
Orosz szorzás	2
Minimum keresés	2
Maximum keresés	2–3
Eukleidész algoritmus	3
Prímszámok	3–4
Fibonacci-számok	4
Háromszög	4–5
Fordított szám	5
Törztényezők	5–6
Prímszámvizsgálat	6
Konverzió – Számrendszer átalakítás	7
Gyors hatványozás	7–8
Szekvenciális (lineáris) keresés	8
Megszámlálás	8
Minimum- és maximumkiválasztás	8
A Maximum helye	8–9
Kiválogatás	9
Szétválogatás	9
Sorozat halmazzá alakítása	9–10
Sorozatok keresztmetszete	10
Sorozatok egyesítése	11
Sorozatok összefésülése	11
Párok sorszáma egy sorozatban	12
Arány	12
Teljes négyzet	12
Osztályátlagok szétválasztása	12–13
Büvös négyzet	13–14
Polinom értéke adott pontban	14
Polinomok összege	14
Polinomok szorzata	14–15
Buborékrendezés (Bubble-sort)	15
Egyszerű felcseréléses rendezés	15–16
Válogatásos rendezés	16
Minimum/maximum kiválasztásra épülő rendezés	16
Beszűrő rendezés	16–17
Leszámláló rendezés	17
Összefésülésen alapuló rendezés	17–18
Gyorsrendezés (QuickSort)	18
Szavak sorrendjének megfordítása	18
Faktoriális	18–19
Számjegyösszeg	19
k elemű részhalmazok	19
Konverzió	19
Az { 1, 2, ..., n } halmaz minden részhalmaza	20
Kamatos kamatok kifrása	20
Általános backtracking	20–21
Általános rekurzív backtracking	21

Elhelyezni 8 királynőt a sakktáblán	21–22
Zárójelek	22
Játékok dobozva való elhelyezésének kiírása	22–23
X pénzösszeg kifizetése n bankjegy segítségével	23–24
X Összeg kifizetése, minimum számú bankjeggyel.....	24
Általános Divide Et Impera	24
Szorzat (DivImp).....	25
Minimumszámolás (DivImp)	25
Hatványozás (DivImp)	25–26
Bináris keresés (DivImp).....	26
Általános mohó (Greedy) algoritmus	26
Összeg (Greedy)	26–27
Hátizsák probléma (Greedy).....	27
Összegkifizetés legkevesebb számú bankjeggyel (Greedy).....	27–28
A C nyelv elemei	28–34
Azonosítók	28
A programok felépítése	28–29
Az alapértelmezett egyszerű típusok	29–30
Változók	30–31
Konstansok (állandók).....	32
Adatok beolvasása és kiírása	32–34
Egyszerű programok készítése	34–36
Téglalap területe és kerülete	35
Inkrementáló és dekrementáló operátorok.....	36
Bitműveletek	36
A C nyelv utasításai	37–48
A kifejezés utasítás	37
Az összetett utasítás	37
Feltételes utasítások	37–39
Az if utasítás	37
Valós szám abszolút értéke	38
Páros – páratlan számok megállapítása.....	38
Nagyobbik szám kiválasztása	38–39
Nagybetű, kisbetű vagy szám.....	39
A switch utasítás.....	39–42
Aritmetikai műveletek	40
Az év hányadik napja.....	41–42
Ciklus utasítások	42–48
Az előtesztelő ciklus	42
A hátultesztelő ciklus	43
A számlálós ciklus	43
Szám számjegyeinek száma	43–44
Törzstényezőkre való bontás.....	44
Legnagyobb közös osztó (Eukleidész algoritmus)	44–45
P számrendszerből q számrendszerbe	45–46
Faktoriális	46
Prímszám vizsgálat	47
Fibonacci sorozat n-dik eleme	47
Polinom értéke egy x pontban (Horner-séma)	48
A tömbök.....	48–59
A tömb fogalma.....	48
Egydimenziós tömbök	48–54
Számsorozat fordított sorrendben	49
Két polinom szorzata	49–50
Kezdőértékkel rendelkező egydimenziós tömbök.....	50

Többdimenziós tömbök	50
Tömb szimmetriája	50–51
Mátrix szorzata vektorral	51–52
Két matrix szorzata	52–53
Mátrix inverze	53–54
Karaktertömbök	55–56
Karakterláncok beolvasása a standard bemenetről	55–56
Karakterláncok kiírása a standard kimenetre	56
Karakterlánc konverziós műveletek	56–58
Numerikus értékek karakterláncná alakítása	56
Karakterlánc numerikus értékké alakítása	56
Más konvertáló függvények	56
Karakterlánc kezelő függvények	56–58
Betűk frekvenciája egy sorban	58
Dinamikus tömbök	58–59
Vektor páros elemeinek összege	58–59
Mátrix páratlan elemeinek összege	59
Az előfeldolgozó parancsok	60–62
Állományok beillesztése	60
Makrók	60–61
Feltételes fordítás makró létezésétől függően	61
Feltételes fordítás makró nem-létezésétől függően	61
Feltételes fordítás	61–62
Fordítási hibahízenet generálása	62
Függvények	62–66
A függvény fogalma	62
Függvény deklarációja	62–63
Függvény definíciója	63
Paraméterátadás	63–64
Tömb paraméterek	64
Faktoriális	64–65
Goldbach-feltétel	65–66
Legnagyobb közös osztó	66
Bonyolultabb adatszerkezetek	67–70
Struktúrák	67–68
Kezdőértékkel rendelkező struktúrák	68
Tanuló adatai	68–69
Saját típusok definiálása	70
Állományok	70–76
Állomány megnyitása	71
Állomány bezárása	71
Írás állományba	71
Karakter írása állományba	71–72
Karakterlánc írása állományba	72
Formázott írás állományba	72
Olvasás állományból	72–76
Karakter olvasása állományól	72
Karakterlánc olvasása állományból	72–73
Formázott olvasás állományból	73
Állomány végének az ellenőrzése	73
Pozicionálás az állományban	73–74
Szöveges állomány feldolgozása (feladat1)	74
Szöveges állomány feldolgozása (feladat2)	74–75
Szöveges állomány feldolgozása (feladat3)	75–76
Klasszikus algoritmusok	76–83

Kereső algoritmusok.....	76–78
Lineáris keresés.....	76–77
Bináris keresés.....	77–78
Rendező algoritmusok.....	78–80
Buborekrendezés.....	78
Minimumkiválasztásos rendezés.....	79
Beszúrásos rendezés.....	79–80
Összefésztülések.....	80–83
Összefésztülés strázsa nélkül.....	80–81
Összefésztülés strázásával (ütközővel).....	82–83
Rekurzió.....	83–92
Közvetlen rekurzió.....	83–88
Szó betűi fordított sorrendben.....	83–84
Faktoriális.....	84
Fibonacci sorozat n-edik eleme.....	84
Legnagyobb közös osztó.....	85
Ackermann függvény.....	85
X^n -diken kiszámítása.....	86
Tizes számrendszerből való átírás p számrendszerbe.....	86–87
Az első n páratlan természetes szám összege.....	87
Természetes szám számjegyeinek összege.....	87–88
N természetes szám összege.....	88
Tömbök rekurzívan.....	88–90
Számsorozat negatív elemeinek száma.....	88–89
Szám előfordulása egy tömbben.....	89
Páros elemek összege.....	90
Rekurzív szerkezetű eredményt igénylő feladatok.....	90–92
N hosszúságú megadott karakterlánc.....	90–91
Természetes szám partíciói.....	91–92
Az „Oszd meg és uralkodj” módszer.....	92–97
Sorozat legnagyobb elem.....	92–93
N szám szorzata.....	93–94
Bináris keresés.....	94
Hanoi tornyok.....	95
QuickSort.....	95–96
MergeSort.....	96–97
Kombinatorikai feladatok.....	98–100
Permutációk.....	98
Variációk.....	98–99
Kombinációk.....	99–100
A dinamikus adatszerkezetek.....	100–104
Szimplán láncolt lista.....	101–104
Lista létrehozása.....	101
Lista elemeinek a kiírása.....	101
Egy új elem beszúrása a listába.....	101–102
Egy elem törlése a listából.....	102
Megoldott listás feladat.....	102–104

Algoritmusok - pszeudókód

Abszolút érték

Határozzuk meg és írjuk ki adott valós szám abszolút értékét!

Algoritmus Abszolút_érték(x, mod):

Ha $x \geq 0$ **akkor** { bemeneti adat: x , kimeneti adat: mod }

$mod \leftarrow x$

különb

$mod \leftarrow -x$

vége(ha)

Vége(algoritmus)

Hányados ismételt kivonással

Számítsuk ki két természetes szám egész hányadosát ismételt kivonásokkal!

Algoritmus Osztas($a, b, hányados$):

$hányados \leftarrow 0$ { bemeneti adatok: a, b , kimeneti adat: $hányados$ }

Amíg $a \geq b$ **végezd el:**

$hányados \leftarrow hányados + 1$

$a \leftarrow a - b$

vége(amíg)

Vége(algoritmus)

Legnagyobb közös osztó

Számítsuk ki két természetes szám legnagyobb közös osztóját!

Algoritmus Eukleidész($a, b, loko$):

Ismételd { bemeneti adatok: a, b , kimeneti adat: $loko$ }

$r \leftarrow \text{maradék}[a/b]$ { kiszámítjuk az aktuális maradékot }

$a \leftarrow b$ { az osztandót felülírjuk az osztóval }

$b \leftarrow r$ { az osztót felülírjuk a maradékkal }

ameddig $r = 0$ { amikor a maradék 0, véget ér az algoritmus }

$loko \leftarrow a$ { loko egyenlő az utolsó osztó értékével }

Vége(algoritmus)

Páros számok szűrése

Számoljuk meg n beolvasott szám közül a páros számokat!

{ bemeneti adat: n és a számok, kimeneti adat: db , a páros számok száma }

Algoritmus Páros(n, db):

$db \leftarrow 0$

Minden $i=1, n$ **végezd el:**

Be: szám

Ha szám *páros* **akkor**

$db \leftarrow db + 1$

vége(ha)

vége(minden)

Vége(algoritmus)

Palindrom számok

Döntsük el egy adott számról, hogy palindromszám-e vagy sem!

Algoritmus Palindrom(szám,válasz):

másolat \leftarrow szám {bemeneti adat: *szám*, kimeneti adat: *válasz*}

újszám \leftarrow 0

Amíg szám > 0 **végezd el:**

számjegy \leftarrow maradék[szám/10]

újszám \leftarrow újszám*10 + számjegy

szám \leftarrow [szám/10]

vége(amíg)

válasz \leftarrow újszám = másolat

{ha újszám = másolat, akkor válasz értéke igaz}

{ha újszám \neq másolat, akkor válasz értéke hamis}

Vége(algoritmus)

Orosz szorzás

Legyen $a, b \in \mathbb{N}^*$. Számítsuk ki a és b szorzatát!

Algoritmus Orosz_szorzás(a,b,p):

$x \leftarrow a$

{bemeneti adatok: a, b}

$y \leftarrow b$

{kimeneti adat: p}

$p \leftarrow 0$

Amíg $x > 0$ **végezd el:**

{ $xy + p = ab$ (*)}

Ha x páratlan **akkor**

$p \leftarrow p + y$

vége(ha)

$x \leftarrow [x/2]$

$y \leftarrow y + y$

vége(amíg)

Vége(algoritmus)

Minimum keresés

Határozzuk meg egy n elemű sorozat minimumát!

Algoritmus Minimum(n, a, \min):

$\min \leftarrow a_1$

Minden $i=2, n$ **végezd el:**

Ha $a_i < \min$ **akkor**

$\min \leftarrow a_i$

vége(ha)

vége(minden)

Vége(algoritmus)

Maximum keresés

Írjuk ki három, páronként különböző valós szám közül a legnagyobbat!

Algoritmus Maximum(a,b,c):

Há $(a > b)$ és $(a > c)$ **akkor**

{bemeneti adatok: a, b, c}

Ki: 'A legnagyobb: ', a

vége(ha)

Ha $(b > c)$ **és** $(b > a)$ **akkor**

Ki: 'A legnagyobb: ', b

vége(ha)

Ha $(c > a)$ **és** $(c > b)$ **akkor**

Ki: 'A legnagyobb: ', c

vége(ha)

Vége(algoritmus)

Eukleidész algoritmusa

Határozzuk meg két adott természetes szám legnagyobb közös osztóját (*lnko*) és legkisebb közös többszörösét (*lkkt*) *Eukleidész algoritmusával!*

Algoritmus Eukleidész(a,b,lnko,lkkt):

$x \leftarrow a$ {bemeneti adatok: a, b, kimeneti adatok: lnko, lkkt}

$y \leftarrow b$

Amíg $a \neq b$ **végezd el:**

Ha $a > b$ **akkor**

$a \leftarrow a - b$

különb

$b \leftarrow b - a$

vége(ha)

vége(amíg)

$lnko \leftarrow a$

$lkkt \leftarrow [x*y/lnko]$

Vége(algoritmus)

Prímszámok

Adva van egy nullától különböző természetes szám (n). Írjunk algoritmust, amely eldönti, hogy az adott szám prímszám-e vagy sem!

Algoritmus Prímek:

Be: határ {határ-nál kisebb számokat fogunk vizsgálni}

Ki: 2

$n \leftarrow 3$

Amíg $n < \text{határ}$ **végezd el:**

prím $\leftarrow igaz$

osztó $\leftarrow 3$

négyzetgyök $\leftarrow [\sqrt{n}]$

Amíg $(osztó \leq \text{négyzetgyök})$ **és** prím **végezd el:**

Ha $\text{maradék}[n/osztó] = 0$ **akkor**

prím $\leftarrow hamis$

különb

osztó $\leftarrow osztó + 2$

vége(ha)

vége(amíg)

Ha prím **akkor**

Ki: n

vége(ha)

$n \leftarrow n + 2$

Rendeld meg a teljes, nyomtatott verziót innen:
www.erettsegi-puskak.ro

www.erettsegi-puskak.ro

vagy

```
#if konstans_kifejezés  
... /* ezt generálja ha a konstans kif. értéke nem nulla */  
#endif
```

Ha a `konstans_kifejezés` (amelyet az előfeldolgozó ki tud értékelni `long int` vagy `unsigned long int` típusú adatként, azaz amely nem függ futási időbeli értéktől), nem nulla, akkor az `#if` utáni utasításokat hajtja végre, egyébként az `#else` utániakat.

Fordítási hibaüzenet generálása

Lehetőségünk van hibaüzenetet generálni. Ennek szintaxisa:

```
#error előfeldolgozó tokenek
```

Lehetővé teszi hibaüzenetek generálását, ha valami hibás beállítást tapasztalunk a előfeldolgozás során. Ez az üzenet megjelenik a fordítóprogram szintaktikai hibaüzenetei között.

Függvények

A függvény fogalma

Hogyha nagyobb feladatot kell megoldanunk, akkor azt általában részfeladatokra bontjuk, hogy a megoldás átláthatóbb legyen. Az egyes részfeladatokat külön függvényekként implementálhatunk. A jól megírt függvények gyakran elrejtik a műveletek részleteit a program azon részei előtt, amelyeknek nem is kell tudniuk róluk. Függvényeket használva, az egész program világosabbá válik, és a változtatások is könnyebben elvégezhetőek. A C nyelvet úgy tervezték meg, hogy a függvények hatékonyak és könnyedén használhatók legyenek. A C programok általában sok kis méretű függvényt tartalmaznak.

Ezekután megadhatjuk a függvénynek egy lehetséges definícióját. A függvény (mint minden programozási nyelvben) utasítások egy csoportja, amely általában egy műveletet valósít meg.

A függvény általános alakja:

```
típus név(formális paraméterlista) paraméterdeklaráció  
{  
    lokális deklarációk  
}
```

Függvény deklarációja

A függvényt a használata előtt deklarálni kell. A függvény deklaráció a függvény fejlécével egyezik meg, és pontosvessző zárja. Formája tehát:

```
típus név(formális paraméterlista);
```

Nem kötelező minden függvényt külön deklarálni is.

Példa:

```
| int max(int a, int b);
```

A formális paraméterek az a és a b egész típusú változók.

Függvény definíciója

A függvény definíciója tulajdonképpen a függvény teljes megadását jelenti. Ha a függvény már volt deklarálva a formális paraméterlistát nem kötelező még egyszer megadni. A definíció általános alakja:

```
| típus név([formális paraméterlista])  
{  
| lokális deklarációk  
| }
```

Ha megírtunk egy függvényt, akkor azt használni is kell. Nézzük, hogyan kell egy függvényt meghívni.

Példa (ha a függvény már deklarálva volt):

```
| int max()  
{  
| return a > b ? a : b;  
| }
```

A függvény meghatározza a paraméterként megadott számok közül a nagyobbbat.

Paraméterátadás

A C programozási nyelvben csak az érték szerinti paraméterátadás ismeretes. Az érték szerinti paraméterátadás esetén a formális paramétereknek van címkomponensük a hívott alprogram területén. Az aktuális paraméternek rendelkeznie kell értékkomponenssel a hívó oldalon. Ez az érték meghatározódik a paraméterkiértékelés folyamán, majd átkerül a hívott alprogram területén lefoglalt címkomponensre. A formális paraméter kap egy kezdőértéket, és az alprogram ezzel az értékkel dolgozik a saját területén.

Példa:

Írjunk egy csere nevű függvényt, amely felcseréli két változó tartalmát!

```
| void csere (int x, int y)  
{  
| int seged;  
| seged = x;  
| x = y;  
| y = seged;  
| }
```

A csere(a, b); függvényhívás nem végzi el a cserét, hiszen csak az a és b változó értékét kapta meg. Ahhoz, hogy a cserét elvégezzük, mutatókat használunk. Át kell tehát adni az a és b

változók címét, hogy az értékük ténylegesen megcserélhető legyen: `cseres(&a, &b);`

Ekkor a cseres függvény definíciója:

```
void cseres (int *px, int *py)
{
    int seged;
    seged = *px;
    *px = *py;
    *py = seged;
}
```

Tömb paraméterek

Amikor a tömbnév egy függvénynek adódik át, a függvény valójában a tömb kezdetének címét kapja meg. A hívott függvényen belül a formális paraméter tehát egy címet tartalmazó változó.

A függvénydefinícióban a

```
| int x[];
```

és a

```
| int *x;
```

Lehet így megadni a formális paraméter típusát, hogy

```
| f3(int m[][6]){ ... }
```

vagy

```
| f3(int (*m)[6]){ ... }
```

A zárójelzés szükséges, mert [] magasabb prioritású művelet, mint a * művelet. Vagyis

```
| int (*m)[6];
```

egy a pointer-t deklarál, ami egy tömbre mutat és a tömb 6 egészéből áll, és

```
| int *m[6];
```

egy a tömböt deklarál, ami 6 pointerből áll és a pointerek egészekre mutatnak.

Faktoriális

Írjunk programot, amely beolvasson egy n számot a $[0,12]$ intervallumból és kiszámítja az $n!$ -t!

A megoldásban a faktoriális egy függvény segítségével számíttjuk ki. Beolvasáskor ellenőrizzük az adatok helyességét. Ha hibás adatot olvasunk be, akkor kilépünk a programból.

```
| #include <stdio.h>
```

```

#include <conio.h>

long faktor (int n);           /* a függvény deklarációja */

main()
{
    int n;
    printf("Kerem az n-et:");
    if (scanf("%d", &n) != 1 || n < 0 || n > 12)
    {
        printf("Hibas adat!");
        getch();
        exit(1);
    }
    printf ("\n%d faktorialisa: %ld",n, faktor(n));
    getch();
}

long faktor (int n)
{
    long fakt;
    int i;
    for (fakt = 1, i = 2; i <= n; fakt *= i++);
    return fakt;
}

```

Goldbach-feltétel

Írjunk programot, amely adott, 2-nél szigorúan nagyobb páros számot két prímszám összegeként ír fel (Goldbach-feltétel ellenőrzése)!

Tekintjük rendre az összes n -nél kisebb páratlan számot. Amikor találunk egy d prímszámot úgy, hogy $n-d$ is prím, akkor kiírjuk a két számot. A prímvizsgálatot egy függvény segítségével végezzük.

```

#include <stdio.h>
#include <conio.h>
#include <math.h>

int prim(int x)
{
    int d, negyzetgyok;
    if(!(x % 2))
        return x == 2;
    d = 3;
    negyzetgyok = floor(sqrt(x));
    while (d < negyzetgyok)
        if (!(x % d))
            return 0;
        else
            d += 2;
    return 1;
}

```

```

main()
{
    int n, d = 3;
    printf("n = ");
    scanf("%d", &n);
    if (n % 2 || n < 4)
    {
        printf("A szam nem paros vagy kisebb mint negy\n");
        return;
    }
    if (prim(n - 2))
    {
        printf("%d = %d + %d\n", n, 2, n-2);
        return;
    }
    while (1)
    {
        if (prim(d) && prim(n - d))
        {
            printf("%d = %d + %d\n", n, d, n-d);
            return;
        }
        else
            d += 2;
        getch();
    }
}

```

Legnagyobb közös osztó

Írjunk programot, amely meghatározza két szám legnagyobb közös osztóját!

```

#include <stdio.h>
#include <conio.h>

int luko(int m, int n)
{
    while (m != n)
        if (m > n)
            m -= n;
        else
            n -= m;
    return m;
}

main()
{
    int a, b;
    printf("a = ");
    scanf("%d", &a);
    printf("b = ");
    scanf("%d", &b);
    printf("A ket szam l.n.k.o.-ja: %d\n", luko(a, b));
    getch();
}

```

Rendeld meg a teljes, nyomtatott verziót innen:
www.erettsegi-puskak.ro

www.erettsegi-puskak.ro

```

    }
}

void main()
{
    printf("n = ");
    scanf("%d", &n);
    particio(0, n);
    getch();
    return;
}

```

Az „Oszd meg és uralkodj” módszer

Ezt a módszert sikeresen lehet alkalmazni az informatikában. A módszer lényege az, hogy a feladatot *egymástól független* részfeladatokra bontjuk, amelyeket az eredeti feladathoz hasonlóan oldunk meg, de kisebb méretű adatok esetében.

A módszer lépéseit a következőképpen foglalhatjuk össze:

- A feladatot kettő vagy több, hasonló egymástól független jellegű részfeladatra bontjuk. A részfeladatok lehetnek *elemi* vagy *nem elemi* részfeladatok.
- Az elemi részfeladatokat megoldjuk, a nem elemieket pedig újabb elemi, vagy nem elemi részfeladatokra bontjuk. Ezt a műveletet addig folytatjuk, ameddig elemi részfeladatokhoz jutunk.
- Az eredményt úgy kapjuk, hogy az egyes részfeladatokat a felosztás fordított sorrendjében összerakjuk, összekombináljuk.

Sorozat legnagyobb elem

Határozzuk meg n egész szám közül a legnagyobbat az oszd meg és uralkodj módszerrel!

```

#include <stdio.h>
#include <conio.h>

int x[100], n, i;

int maximum(int bal, int jobb)
{
    int max1, max2, kozepe;
    if (bal == jobb)
        return x[bal];
    else
        if (jobb - bal == 1)
            if (x[bal] < x[jobb])
                return x[jobb];
            else
                return x[bal];
        else
            {
                kozepe = (bal + jobb) / 2;
                max1 = maximum(bal, kozepe);

```

```

    max2 = maximum(kozepe + 1, jobb);
    if (max1 < max2)
        return max2;
    else
        return max1;
    }
}

void main()
{
    printf("n = ");
    scanf("%d", &n);
    printf("Adjuk meg a számokat!\n");
    for (i = 0; i < n; i++)
    {
        printf("Az %d .elem: ", i);
        scanf("%d", &x[i]);
    }
    printf("A legnagyobb szám : %d", maximum(0, n - 1));
    getch();
    return;
}

```

N szám szorzata

Számítsuk ki n valós szám szorzatát oszd meg és uralkodj módszerrel!

```

#include <stdio.h>
#include <conio.h>

int x[100], n, i;

int szorzas(int bal, int jobb)
{
    int sz1, sz2, kozepe;
    if (bal == jobb)
        return x[bal];
    else
        if (jobb - bal == 1)
            return x[bal] * x[jobb];
        else
        {
            kozepe = (bal + jobb) / 2;
            sz1 = szorzas(bal, kozepe);
            sz2 = szorzas(kozepe + 1, jobb);
            return sz1 * sz2;
        }
}

void main()
{
    printf("n = ");
    scanf("%d", &n);
    printf("Adjuk meg a számokat!\n");
    for (i = 0; i < n; i++)

```

```

{
    printf("Az %d .elem: ", i);
    scanf("%d", &x[i]);
}
printf("A szamok szorzata : %d", szorzas(0, n - 1));
getch();
return;
}

```

Bináris keresés

Olvassunk be egy rendezett számsorozatot, és keressünk meg benne egy beolvasott számot! Ha megtaláltuk írjuk ki a pozícióját. Alkalmazzuk a bináris keresés módszerét!

```

#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

int x[100], n, i, ker;

void bin_kereses(int bal, int jobb)
{
    int kozepe;
    if (bal > jobb)
    {
        printf("%d nincs az adott sorozatban", ker);
        exit;
    }
    kozepe = (bal + jobb) / 2;
    if (ker == x[kozepe])
        printf("%d a(z) %d h. talalhato", ker, kozepe + 1);
    else
        if (ker < x[kozepe])
            bin_kereses(bal, kozepe - 1);
        else
            bin_kereses(kozepe + 1, jobb);
}

void main()
{
    printf("n = ");
    scanf("%d", &n);
    printf("Adjuk meg a sorozatot!\n");
    for (i = 0; i < n; i++)
    {
        printf("Az %d .elem: ", i);
        scanf("%d", &x[i]);
    }
    printf("A keresett szam: ");
    scanf("%d", &ker);
    bin_kereses(0, n - 1);
    getch();
    return;
}

```

Hanoi tornyok

Adva van három rúd: A, B és C. Az A rúdon induláskor n különböző átmérőjű lyukas korong található, az átmérők szerint csökkenő sorrendben. Írjuk ki az összes lehetséges módját annak, ahogyan a korongokat át lehet helyezni az A rúdról a B-re, ugyanolyan sorrendben, ahogyan az A-n helyezkedtek el! Az áthelyezés közben fel lehet használni a C rudat. Egy mozgás alkalmával csak egy korongot lehet áthelyezni, és csak kisebb átmérőjű korongot helyezhetünk egy nagyobb átmérőjű korongra.

```
#include <stdio.h>
#include <conio.h>

int n;

void hanoi(int n, char a, char b, char c)
{
    if (n == 1)
        printf("%c - %c\n", a, b);
    else
    {
        hanoi(n - 1, a, c, b);
        printf("%c - %c\n", a, b);
        hanoi(n - 1, c, b, a);
    }
}

void main()
{
    printf("n = ");
    scanf("%d", &n);
    printf("A mozgások: \n");
    hanoi(n, 'A', 'B', 'C');
    getch();
    return;
}
```

QuickSort

Rendezzünk növekvő sorrendbe n egész számot, a QuickSort algoritmussal!

```
#include <stdio.h>
#include <conio.h>

int x[100], n, i, m;

int elvalaszto_helye(int bal, int jobb)
{
    int elvalaszto, seged, i, j;
    elvalaszto = x[bal];
    i = bal - 1;
    j = jobb + 1;
    do
    {
        do
```

Rendeld meg a teljes, nyomtatott verziót innen:
www.erettsegi-puskak.ro

www.erettsegi-puskak.ro