1. Feladat: beolvas két számot úgy, hogy a-ba kerüljön a nagyobb

```
#include<stdio.h>
main()
{
    int a, b;
    printf( "a=" );  scanf( "%d", &a );
    printf( "b=" );  scanf( "%d", &b );
    if( a< b )
    {
            inttmp = a;
            a = b;
            b = tmp;
    }
}
```

2. Feladat: pontszám alapján eldönti, hogy sikeres volt-e a dolgozat - 60%

```
#include<stdio.h>
main()
{
    intelerheto, pontszam;
    printf( "elérhető=" );  scanf( "%d", &elerheto );
    printf( "pontszám=" );  scanf( "%d", &pontszam );
    if( 100 * pontszam>= 60 * elerheto )
            puts( "A dolgozat sikeres." );
    else
            puts( "A dolgozat sikertelen." );
}
```

3. Feladat: a beolvasott évszámról eldönti, hogy szökőév-e

```
#include<stdio.h>
main()
{
 intevszam;
 printf( "Évszám: " );  scanf( "%d", &evszam );
 if( evszam % 4 == 0 &&evszam % 100 != 0 || evszam % 400 == 0 )
     puts( "Szökőév." );
 else
     puts( "Nem szökőév." );
}
```

4. Feladat: három szakaszról eldönti, hogy szerkeszthető-e háromszög belőlük, ha igen, kiszámítja annak területét

```
#include<math.h>
#include<stdio.h>
main()
{
     doublea, b, c;
     printf( "a=" );  scanf( "%lf", &a );
     printf( "b=" );  scanf( "%lf", &b );
     printf( "c=" );  scanf( "%lf", &c );
     if( a> 0 && b > 0 && c > 0 && a + b > c && a + c > b && b + c > a )
     {
               double s = ( a + b + c ) / 2;
          puts( "Szerkeszthető háromszög a szakaszokból." );
          printf( "A háromszög területe: %lf.\n",
          sqrt( s * ( s - a ) * ( s - b ) * ( s - c ) ) );
     }
```

```
    else
        puts( "Nem szerkeszthető háromszög a szakaszokból." );
}
```

5. Írjunk programot, amely értékel egy dolgozatot a rá adott pontszám alapján! Az értékelés az alábbiak alapján történjen:
   - 0-42: elégtelen,
   - 43-57: elégséges,
   - 58-72: közepes,
   - 73-87: jó,
   - 88-100: jeles.

A feladatot if - elseif –else szerkezettel valósítsuk meg!

```
#include <stdio.h>

main()
{
  int pont;
  printf( "Pontszám: " );  scanf( "%d", &pont );
  if ( pont < 0 || pont > 100 )
    puts( "Érvénytelen pontszám." );
  else if ( pont <= 42 )
    puts( "Elégtelen." );
  else if ( pont <= 57 )
    puts( "Elégséges." );
  else if ( pont <= 72 )
    puts( "Közepes." );
  else if ( pont <= 87 )
    puts( "Jó." );
  else
    puts( "Jeles." );
}
```

6. Írjunk programot, amely az *a, b, c* értékek ismeretében képernyőre írja az $ax^2+bx+c=0$ másodfokú egyenlet megoldásait!

```c
#include <math.h>
#include <stdio.h>

main()
{
  double a, b, c;
  printf( "a=" );  scanf( "%lf", &a );
  printf( "b=" );  scanf( "%lf", &b );
  printf( "c=" );  scanf( "%lf", &c );
  if ( a == 0 )
  {
    if ( b == 0 )
    {
      if ( c == 0 )
        puts( "Az egyenlet azonosság. Megoldása minden valós szám." );
      else
        puts( "Az egyenlet ellentmondásos, nincs megoldása." );
    }
    else
    {
      puts( "Az egyenlet elsőfokú." );
      printf( "Megoldása: x=%lf\n", -c / b );
    }
  }
  else
  {
    double diszkriminans = b * b - 4 * a * c;
    if ( diszkriminans < 0 )
      puts( "Az egyenletnek a valós számok körében nincs megoldása." );
    else if ( diszkriminans == 0 )
    {
      puts( "Az egyenletnek két egybeeső megoldása van." );
      printf( "Ezek a következők: x1 = x2 = %lf\n", -b / ( 2 * a ) );
    }
    else
    {
      puts( "Az egyenletnek két különböző megoldása van." );
      printf( "Ezek a következők: x1 = %lf, x2 = %lf\n",
              ( -b + sqrt( diszkriminans ) ) / ( 2 * a ),
              ( -b - sqrt( diszkriminans ) ) / ( 2 * a ) );
    }
  }
}
```

7. Írjunk programot, amely kiírja egy dolgozat szöveges értékelését az érdemjegy alapján! Többirányú elágaztató utasítást használjunk!

```c
#include <stdio.h>

main()
{
  int jegy;
  printf( "Érdemjegy: " );  scanf( "%d", &jegy );
  switch ( jegy )
  {
    case 1: puts( "elégtelen" ); break;
    case 2: puts( "elégséges" ); break;
    case 3: puts( "közepes" );   break;
    case 4: puts( "jó" );        break;
    case 5: puts( "jeles" );     break;
  }
}
```

8. Írjunk programot, amely for ciklussal kiírja az első tíz természetes számot és azok négyzetét!

```c
#include <stdio.h>

main()
{
  int szam;
  for ( szam = 0; szam < 10; ++szam )
    printf( "%d %d\n", szam, szam * szam );
}
```

9. Írjunk programot, amely while ciklussal karaktereket olvas be standard inputról, és átmásolja azokat a standard outputra!

```c
#include <stdio.h>

main()
{
  int ch;
  while ( ( ch = getchar() ) != EOF )
    putchar( ch );
}
```

10. Írjunk programot, amely kiszámítja egy 0 és 12 közötti egész szám faktoriálisát (azért csak 12-ig, mert a 12 faktoriálisa még tárolható egy unsigned long int típusban)!

```c
#include <stdio.h>

main()
{
  unsigned szam, i;
  unsigned long fakt = 1;
  printf( "Szám: " );  scanf( "%u", &szam );
  for ( i = 2; i <= szam; ++i )
    fakt *= i;
  printf( "%u faktoriálisa: %lu.\n", szam, fakt );
}
```

11. Írjunk programot, mely for ciklussal kiszámítja a Fibonacci-sorozat n-edik elemét, ahol n egy nem túl nagy természetes szám!

```c
#include <stdio.h>

main()
{
  unsigned szam, i;
  long akt = 1, elozo = 1;
  printf( "Szám: " );  scanf( "%u", &szam );
  for ( i = 3; i <= szam; ++i )
  {
    long uj = elozo + akt;
    elozo = akt;
    akt = uj;
  }
  printf( "A Fibonacci-sorozat %u. eleme: %ld.\n", szam, akt );
}
```

```c
#include <stdio.h>

main()
{
  unsigned szam, i;
  long akt = 1, elozo = 1;
  printf( "Szám: " );  scanf( "%u", &szam );
  for ( i = 3; i <= szam; ++i )
  {
    akt += elozo;
    elozo = akt - elozo;
  }
  printf( "A Fibonacci-sorozat %u. eleme: %ld.\n", szam, akt );
}
```

12. Írjunk programot, amelywhile ciklussal 0 végjelig olvas be egész számokat, és mindegyikről eldönti, hogy páros-e vagy páratlan!

```c
#include <stdio.h>

main()
{
  int szam;
  puts( "Kérem a számokat:" );
  scanf( "%d", &szam );
  while ( szam )
  {
    if ( szam % 2 == 0 )
      printf( "%d páros.\n", szam );
    else
      printf( "%d páratlan.\n", szam );
    scanf( "%d", &szam );
  }
}
```

13.Írjunk programot, amelywhile ciklussal meghatározza két pozitív egész szám legnagyobb közös osztóját!

```c
#include <stdio.h>

main()
{
  unsigned a, b;
  printf( "a = " );  scanf( "%u", &a );
  printf( "b = " );  scanf( "%u", &b );
  while ( a != b )
    if ( a > b )
      a -= b;
    else
      b -= a;
  printf( "A legnagyobb közös osztó: %u.\n", a );
}
```

```c
#include <stdio.h>

main()
{
  unsigned a, b, r;
  printf( "a = " );  scanf( "%u", &a );
  printf( "b = " );  scanf( "%u", &b );
  while ( r = a % b )
  {
    a = b;
    b = r;
  }
  printf( "A legnagyobb közös osztó: %u.\n", b );
}
```

14. Írjunk programot, mely meghatározza egy billentyűzetről beolvasott természetes szám prímtényezős felbontását!

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

main()
{
  int szam;
  printf( "Szám: " );  scanf( "%d", &szam );
  if ( szam == 1 )
    puts( "1" );
  else
  {
    while ( szam != 1 )
    {
      int oszto = 2;
      for ( ; oszto <= sqrt( szam ) && szam % oszto != 0; ++oszto )
        ;
      if ( oszto > sqrt( szam ) )
        oszto = szam;
      printf( "%d ", oszto );
      szam /= oszto;
    }
    putchar( '\n' );
  }
}
```

15. Írjunk programot, amely a billentyűzetről karaktereket olvas mindaddig, amíg azok az angol abc betűi, majd a beolvasás után kiírja, hogy hány volt ezek közül kisbetű!

```
#include <stdio.h>

main()
{
  int szamlalo = 0;
  char ch;
  printf( "Kérem a karaktereket:\n" );
  do{
    ch = getc(stdin);

    if (islower(ch)){
      szamlalo++;
    }
  }
  while (isalpha(ch) || ch == '\n');
  printf( "A kisbetűk száma: %d.\n", szamlalo );
}
```

16. Írjunk programot, amely egyesével beolvas egész számokat mindaddig, amíg a számok váltakozó előjelűek, majd kiírja a képernyőre a beolvasott értékek darabszámát! A nulla egy speciális érték, amelyet a negatív számok közé sorolunk, ha előtte pozitív érték szerepel (beleértve a pozitív számok közé sorolt nullát is), illetve a pozitívak közé, ha előtte negatív érték áll (beleértve a negatív számok közé sorolt nullát is). A sorozat elején álló nulla értékek előjele a sorozat első nem nulla értékének előjelétől függ.

```
#include <stdio.h>

main()
{
  enum { SEMLEGES, POZITIV, NEGATIV, VEG } allapot = SEMLEGES;
  int szam, darab = 0;
  do
  {
    ++darab;
    scanf( "%d", &szam );
    switch ( allapot )
    {
      case SEMLEGES:
        if ( szam < 0 )
          allapot = NEGATIV;
        else if ( szam > 0 )
          allapot = POZITIV;
        break;
      case POZITIV:
        allapot = szam > 0 ? VEG : NEGATIV;
        break;
      case NEGATIV:
        allapot = szam < 0 ? VEG : POZITIV;
        break;
    }
  } while ( allapot != VEG );
  printf( "A beolvasott értékek száma: %d.\n", darab );
}
```

17. Írjunk programot, amely a billentyűzetről látható karaktereket olvas mindaddig, amíg a @ karaktert meg nem kapja! A program határozza meg és írja képernyőre a beolvasott különböző karaktereket és azok gyakoriságát!

```c
#include <stdio.h>

main()
{
  int c;
  int gyak[ 256 ] = { 0 };   /* az egész tömböt nullázza */
  while ( ( c = getchar() ) != '@' )
    ++gyak[ c ];
  for ( c = 0; c < 256; ++c )
    if ( gyak[ c ] )
      printf( "%c: %d\n", c, gyak[ c ] );
}
```

18. Írjunk eljárást, mely paraméterként megkap egy tetszőleges méretű, egészeket tartalmazó egydimenziós tömböt! Az eljárás határozza meg a tömbben lévő pozitív, negative és nulla elemek darabszámát!

```c
int pozitiv, negativ, nulla;

void poznegnull( int *t, int meret )
{
  int i;
  pozitiv = negativ = nulla = 0;
  for ( i = 0; i < meret; ++i )
    if ( t[ i ] > 0 )
      ++pozitiv;
    else if ( t[ i ] < 0 )
      ++negativ;
    else
      ++nulla;
}
```

```c
void poznegnull( int *t, int meret, int *poz, int *neg, int *nulla )
{
  int i;
  *poz = *neg = *nulla = 0;
  for ( i = 0; i < meret; ++i )
    if ( t[ i ] > 0 )
      ++*poz;
    else if ( t[ i ] < 0 )
      ++*neg;
    else
      ++*nulla;
}
```

19. Írjunk programot, amely nullától különböző egészeket olvas be a billentyűzetről a 0 végjelig! A program határozza meg és írja a képernyőre azt a három értéket, amelynek átlaga maximális!

```c
#include <stdio.h>

void csokkeno_rendez( int t[], int meret )
{
  int i, j;
  for ( i = meret - 2; i >= 0; --i )
    for ( j = 0; j <= i; ++j )
      if ( t[ j ] < t[ j + 1 ] )
      {
        int seged = t[ j ];
        t[ j ] = t[ j + 1 ];
        t[ j + 1 ] = seged;
      }
}

main()
{
  int *tomb = NULL, meret = 0;
  for ( ; ; )
  {
    tomb = ( int * )realloc( tomb, ( meret + 1 ) * sizeof( int ) );
    scanf( "%d", &tomb[ meret ] );
    if ( tomb[ meret ] == 0 )
      break;
    ++meret;
  }
  if ( meret < 3 )
    puts( "Nincs három érték." );
  else
  {
    csokkeno_rendez( tomb, meret );
    printf( "%d, %d, %d\n", tomb[ 0 ], tomb[ 1 ], tomb[ 2 ] );
  }
  free( tomb );
}
```

```c
#include <stdio.h>

#define DARAB 3

main()
{
  int tomb[ DARAB ], meret = 0, szam;
  for ( ; ; )
  {
    scanf( "%d", &szam );
    if ( szam == 0 )
      break;
    if ( meret < DARAB )
      tomb[ meret++ ] = szam;
    else
    {
      int index = tomb[ 0 ] < tomb[ 1 ] ? tomb[ 0 ] < tomb[ 2 ] ?
        0 : 2 : tomb[ 1 ] < tomb[ 2 ] ? 1 : 2;
      if ( szam > tomb[ index ] )
        tomb[ index ] = szam;
    }
  }
  if ( meret < DARAB )
    puts( "Nincs három érték." );
  else
    printf( "%d, %d, %d\n", tomb[ 0 ], tomb[ 1 ], tomb[ 2 ] );
}
```

20. Írjunk programot, amely megoldja a jól ismert ,,nyolc királynő” problémát, azaz elhelye egy sakktáblán nyolc királynőt úgy, hogy azok ne üssék egymást, és kiírja a képernyőre a probléma összes megoldását!

```c
#include <stdio.h>
#include <stdlib.h>

#define N 8
#define HAMIS 0
#define IGAZ  ( !HAMIS )

int tabla[ N ];   /* automatikus nullázás van */

int elofeltetel( int sor, int oszlop )
{
  int i;
  for ( i = 0; i < oszlop; ++i )
    if ( tabla[i] == sor || abs( tabla[i]-sor ) == abs( i-oszlop ) )
      return HAMIS;
  return IGAZ;
}

main()
{
  int index = 0;
  while ( index >= 0 )
  {
    if ( index == N )
    {
      int i;
      for ( i = 0; i < N; ++i )
        if ( i )
          printf( " %c%d", 'a' + i, tabla[ i ] + 1 );
        else
          printf( "%c%d", 'a' + i, tabla[ i ] + 1 );
      putchar( '\n' );
      ++tabla[ --index ];
    }
    while ( tabla[index] < N && !elofeltetel( tabla[index], index ) )
      ++tabla[ index ];
    if ( tabla[ index ] == N )
    {
      tabla[ index ] = 0;
      ++tabla[ --index ];
    }
    else
      ++index;
  }
}
```

```c
#include <stdio.h>
#include <stdlib.h>

#define N 8
#define HAMIS 0
#define IGAZ  ( !HAMIS )

int tabla[ N ];

int elofeltetel( int sor, int oszlop )
{
  int i;
  for ( i = 0; i < oszlop; ++i )
    if ( tabla[i] == sor || abs( tabla[i]-sor ) == abs( i-oszlop ) )
      return HAMIS;
  return IGAZ;
}

void kiralyno( int darab )
{
  int i;
  if ( darab == 0 )
  {
    for ( i = 0; i < N; ++i )
      if ( i )
        printf( " %c%d", 'a' + i, tabla[ i ] + 1 );
      else
        printf( "%c%d", 'a' + i, tabla[ i ] + 1 );
    putchar( '\n' );
  }
  else
  {
    for ( i = 0; i < N; ++i )
      if ( elofeltetel( i, N - darab ) )
      {
        tabla[ N - darab ] = i;
        kiralyno( darab - 1 );
      }
  }
}

main()
{
  kiralyno( N );
}
```

21. Írjunk függvényt, amely egy paraméterként megkapott, egészeket tartalmazó kétdimenziós tomb esetén megadja azon oszlopok számát, amelyekben egy szintén paraméterként megadott értéknél csak nagyobb értékek szerepelnek!

```
int csaknagyobb( int *t, int sor, int oszlop, int ertek )
{
  int j, db = 0;
  for ( j = 0; j < oszlop; ++j )
  {
    int i;
    for ( i = 0; i < sor; ++i )
      if ( t[ i * oszlop + j ] <= ertek )
        break;
    if ( i == sor )
      ++db;
  }
  return db;
}
```

22. Írjunk eljárást, amely paraméterként megkap egy tetszőleges méretű, valósakat tartalmazó kétdimenziós tömböt, és előállít egy olyan egydimenziós tömböt, amely a sorok átlagát tartalmazza!

```
#include <stdlib.h>

double *soratl;

void soratlagok( double *t, int sor, int oszlop )
{
  int i;
  soratl = ( double * )calloc( sor, sizeof( double ) );
  for ( i = 0; i < sor; ++i )
  {
    int j;
    for ( j = 0; j < oszlop; ++j )
      soratl[ i ] += t[ i * oszlop + j ];
    soratl[ i ] /= oszlop;
  }
}
```

23. Írjunk eljárást ,amely egy paraméterként megkapott, egészeket tartalmazó kédimenziós tomb oszlopait úgy rendezi át, hogy az első sor elemei nagyság szerint csökkenő sorrendben legyenek!

```
void csokelsosor( int *t, int sor, int oszlop )
{
  int korlat = oszlop - 1, utolsocsere;
  do
  {
    int j;
    utolsocsere = -1;
    for ( j = 0; j < korlat; ++j )
      if ( t[ j ] < t[ j + 1 ] )
      {
        int i;
        /* az oszlopok minden elemét cseréljük */
        for ( i = 0; i < sor; ++i )
        {
          int seged = t[ i * oszlop + j ];
          t[ i * oszlop + j ] = t[ i * oszlop + j + 1 ];
          t[ i * oszlop + j + 1 ] = seged;
        }
        utolsocsere = j;
      }
    korlat = utolsocsere;
  } while ( utolsocsere != -1 );
}
```

RENDEZÉSEK

24. Írjunk eljárást, amely növekvő sorrenbe rendezi a paraméterként megkapott, egészeket tartalmazó tömböt maximumkiválasztásos rendezéssel!

```
void csere( int tomb[], int i, int j )
{
  int seged = tomb[ i ];
  tomb[ i ] = tomb[ j ];
  tomb[ j ] = seged;
}

void maxkival( int tomb[], int meret )
{
  int j;
  for ( j = meret - 1; j > 0; --j )
  {
    int max = j, i;
    for ( i = 0; i < j; ++i )
      if ( tomb[ i ] > tomb[ max ] )
        max = i;
    csere( tomb, max, j );
  }
}
```

25. Írjunk eljárást, amely növekvő sorrenbe a paraméterként megkapott, egészeket tartalmazó tömböt beszúrásos rendezéssel!

```
void beszurasos( int tomb[], int meret )
{
  int j;
  for ( j = 1; j < meret; ++j )
  {
    int kulcs = tomb[ j ], i = j - 1;
    while ( i >= 0 && tomb[ i ] > kulcs )
    {
      tomb[ i + 1 ] = tomb[ i ];
      --i;
    }
    tomb[ i + 1 ] = kulcs;
  }
}
```

26. Írjunk eljárást, amely növekvő sorrenbe a paraméterként megkapott, egészeket tartalmazó tömböt buborékrendezéssel!

```
void csere( int tomb[], int i, int j )
{
  int seged = tomb[ i ];
  tomb[ i ] = tomb[ j ];
  tomb[ j ] = seged;
}

void buborek1( int tomb[], int meret )
{
  int i, j;
  for ( i = meret - 1; i > 0; --i )
    for ( j = 0; j < i; ++j )
      if ( tomb[ j + 1 ] < tomb[ j ] )
        csere( tomb, j, j + 1 );
}
```

```
#define HAMIS 0
#define IGAZ  ( !HAMIS )

void csere( int tomb[], int i, int j )
{
  int seged = tomb[ i ];
  tomb[ i ] = tomb[ j ];
  tomb[ j ] = seged;
}

void buborek2( int tomb[], int meret )
{
  int i, j, voltcsere = IGAZ;
  for ( i = meret - 1; i > 0 && voltcsere; --i )
  {
    voltcsere = HAMIS;
    for ( j = 0; j < i; ++j )
      if ( tomb[ j + 1 ] < tomb[ j ] )
      {
        csere( tomb, j, j + 1 );
        voltcsere = IGAZ;
      }
  }
}
```

27. Írjunk eljárást, amely növekvő sorrenbe a paraméterként megkapott, egészeket tartalmazó tömböt gyorsrendezéssel!

```
void csere( int tomb[], int i, int j )
{
  int seged = tomb[ i ];
  tomb[ i ] = tomb[ j ];
  tomb[ j ] = seged;
}

void gyors( int tomb[], int bal, int jobb )
{
  if ( bal < jobb )
  {
    int also = bal, felso = jobb + 1, kulcs = tomb[ bal ];
    for ( ; ; )
    {
      while ( ++also < felso && tomb[ also ] < kulcs )
        ;
      while ( tomb[ --felso ] > kulcs )
        ;
      if ( also >= felso )
        break;
      csere( tomb, also, felso );
    }
    csere( tomb, felso, bal );
    gyors( tomb, bal, felso - 1 );
    gyors( tomb, felso + 1, jobb );
  }
}
```

```
void csere( int tomb[], int i, int j )
{
  int seged = tomb[ i ];
  tomb[ i ] = tomb[ j ];
  tomb[ j ] = seged;
}

int feloszt( int tomb[], int bal, int jobb )
{
  int kulcs = tomb[ jobb ], hatar = bal - 1, akt;
  for ( akt = bal; akt < jobb; ++akt )
    if ( tomb[ akt ] <= kulcs )
      csere( tomb, ++hatar, akt );
  csere( tomb, hatar + 1, jobb );
  return hatar + 1;
}

void gyors2( int tomb[], int bal, int jobb )
{
  if ( bal < jobb )
  {
    int kozepso = feloszt( tomb, bal, jobb );
    gyors2( tomb, bal, kozepso - 1 );
    gyors2( tomb, kozepso + 1, jobb );
  }
}
```

KERESÉSEK

28. Írjunk függvényt, amely teljes kereséssel megkeresi a paraméterként megkapott tömbben a szintén paraméterként megkapott értéket, és az első olyan elem indexével tér vissza, amelynek értéke a keresett érték! Ha a keresett érték nincs a tömbben, akkor -1-et kell visszaadni!

```
int teljes( int tomb[], int meret, int ertek )
{
  int i;
  for ( i = 0; i < meret && tomb[ i ] != ertek; ++i )
    ;
  return i < meret ? i : -1;
}
```

29. Írjunk függvényt, amely lineáris kereséssel megkeresi a paraméterként megkapott tömbben a szintén paraméterként megkapott értéket, és az első olyan elem indexével tér vissza, amelynek értéke a keresett érték! Ha a keresett érték nincs a tömbben, akkor -1-et kell visszaadni!

```
int linearis( int tomb[], int meret, int ertek )
{
  int i;
  for ( i = 0; i < meret && tomb[ i ] < ertek; ++i )
    ;
  return i < meret && tomb[ i ] == ertek ? i : -1;
}
```

30. Írjunk függvényt, amely bináris kereséssel megkeresi a paraméterként megkapott tömbben a szintén paraméterként megkapott értéket, és az első olyan elem indexével tér vissza, amelynek értéke a keresett érték! Ha a keresett érték nincs a tömbben, akkor -1-et kell visszaadni!

```
int binaris( int tomb[], int meret, int ertek )
{
  int also = 0, felso = meret - 1;
  while ( also <= felso )
  {
    int kozepso = ( also + felso ) / 2;
    if ( tomb[ kozepso ] == ertek )
      return kozepso;
    if ( tomb[ kozepso ] > ertek )
      felso = kozepso - 1;
    else
      also = kozepso + 1;
  }
  return -1;
}
```

```
int binaris_rek( int tomb[], int also, int felso, int ertek )
{
  if ( also <= felso )
  {
    int kozepso = ( also + felso ) / 2;
    if ( tomb[ kozepso ] == ertek )
      return kozepso;
    return tomb[ kozepso ] > ertek ?
           binaris_rek( tomb, also, kozepso - 1 ) :
           binaris_rek( tomb, kozepso + 1, felso );
  }
  return -1;
}
```