Kósa Márk – Pánovics János

Példatár a *Programozás 1* tárgyhoz

Juhász István feladatsoraiból

mobiDIÁK könyvtár

Kósa Márk – Pánovics János

Példatár a *Programozás 1* tárgyhoz

Juhász István feladatsoraiból

mobiDIÁK könyvtár

SOROZATSZERKESZTŐ

Fazekas István

Kósa Márk – Pánovics János

Példatár a *Programozás 1* tárgyhoz

Juhász István feladatsoraiból

Egyetemi jegyzet

első kiadás

mobiDIÁK könyvtár

Debreceni Egyetem
Informatikai Intézet

Lektor

Juhász István
Debreceni Egyetem

mobiDIÁK könyvtár
Debreceni Egyetem
Informatikai Intézet
4010 Debrecen, Pf. 12
http://mobidiak.unideb.hu

A mű a *A mobiDIÁK önszervező mobil portál* (IKTA, OMFB-00373/2003) és a *GNU Iterátor, a legújabb generációs portál szoftver* (ITEM, 50/2003) projektek keretében készült.

Tartalomjegyzék

Előszó

A *Programozás 1* tárgy gyakorlatainak keretében a 2001/2002-es tanévtől kezdve mindhárom képzési formában (nappali, levelező és esti tagozaton) a C programozási nyelvet tanulják hallgatóink. A félév végi „beugrókon” is e nyelven kell kódolniuk a feladataikat. Természetes az igény, hogy mindenki számára hozzáférhető módon rendelkezésre álljon egy feladatgyűjtemény, amely a feladatok szövege mellett tartalmazza azok lehetséges megoldásait is.

Ez a gyűjtemény egy válogatás a tárgy előadójának, Juhász Istvánnak a „beugrókon” íratott feladatsoraiból, az 1994-es nyári vizsgaidőszaktól kezdődően egészen napjainkig. A korai feladatsorok egy konkrét programozási nyelvhez (Turbo Pascalhoz vagy C-hez) kapcsolódtak, míg az újabbak tetszőleges nyelven megoldhatók. Igazodva azonban a tantárgy gyakorlati követelményeihez, ebben a példatárban csak C nyelvű programkódokat adunk közre. A feladatokat témakörönként csoportosítottuk, minden témakörön belül igyekeztünk egy nehézségi sorrendet is felállítani. A legtöbb feladatnak csak egy lehetséges megoldását közöltük, ám ahol több, egymástól valamilyen lényeges szempontból különböző megoldás is létezett, ott azok közül többet is megadtunk. Ugyanakkor az is elképzelhető – és ettől szép a programozás –, hogy az olvasó fog kitalálni az itt leírtaktól eltérő, esetleg egyszerűbb megoldást egyik-másik feladatra.

A példatárban közreadott programkódokat Linux operációs rendszer alatt GNU C fordítóval teszteltük, de bármely ANSI C fordítóval lefordíthatók.

A feladatgyűjtemény végén megtalálható az összes „beugró” feladatsor időrendi sorrendben 1994. május 16-tól kezdve. Ezekből az olvasó képet kaphat arról, hogy milyen nehézségű feladatsorokat kellett megoldaniuk a hallgatóknak az elmúlt években, és egyúttal tesztelheti saját tudását is.

Sikeres felkészülést kívánnak a szerzők, akik a feladatgyűjteményben esetleg benne maradt apróbb hibákért elnézést kérnek, és minden javító szándékú kritikát szívesen vesznek a következő címek valamelyikén: `mkosa@inf.unideb.hu` vagy `panovics@inf.unideb.hu`.

Debrecen, 2004. január 18.

Kósa Márk
Pánovics János

1. fejezet

Példák tematikus csoportosításban

1.1. A legegyszerűbb feladatok

1.1. FELADAT. Írjon **programot**, amely 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**!

Egy tetszőleges karakterről a `ctype.h`-ban definiált `isprint()` függvény segítségével tudjuk eldönteni, hogy látható-e (nyomtatható-e). Azonban mivel most kizárólag látható karaktereket olvasunk, ezért ez az ellenőrzés elhagyható. Így egy lehetséges megoldás a következő:

```
#include <stdio.h>

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

1.2. FELADAT. Írjon **programot**, amely **billentyűzetről** egész értékeket olvas be a **0** végjelig. A program írja **képernyőre** azokat az értékeket, amelyek **megegyeznek** az **előző két** érték **összegével**.

```
#include <stdio.h>

#define FALSE 0
#define TRUE  !FALSE

main()
{
  int t[ 3 ], idx = -1, megvolt = FALSE;
  for ( ; ; )
  {
    scanf( "%d", &t[ idx = ( idx + 1 ) % 3 ] );
    if ( !t[ idx ] )
      break;
    if ( idx == 2 )
      megvolt = TRUE;
    if ( megvolt && t[ idx ] == t[ ( idx + 1 ) % 3 ] + t[ ( idx + 2 ) % 3 ] )
      printf( "%d\n", t[ idx ] );
 }
}
```

1.3. FELADAT. Írjon **programot**, amely nullától különböző **egész** értékeket olvas be a **billentyűzetről** a **0** végjelig. A program határozza meg és írja **képernyőre** azt a **három** értéket, amelynek **átlaga maximális**.

A feladat megfogalmazása csalafinta, gyakorlatilag a három legnagyobb értéket kell kiválasztani a begépelt számok közül, hiszen ezek átlaga lesz maximális. Háromnál kevesebb beolvasott érték esetén a programok csak egy figyelmeztető üzenetet írnak a képernyőre.

(a) Az első megoldás tárolja az összes beolvasott értéket, majd a 0 érték beolvasása után csökkenő sorrendbe rendezi őket, és kiírja közülük az első hármat (ha van legalább három érték).

```
#include <stdio.h>

void rendez_csok( 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
  {
    rendez_csok( tomb, meret );
    printf( "%d, %d, %d\n", tomb[ 0 ], tomb[ 1 ], tomb[ 2 ] );
  }
  free( tomb );
}
```

(b) A második megoldás csak a három legnagyobb értéket tárolja úgy, hogy mindig a legkisebb tárolt értéket írja felül, ha egy nála nagyobb értéket olvas.

```
#include <stdio.h>

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

(c) A harmadik megoldás a második megoldáshoz hasonlóan a három legnagyobb értéket tárolja, amelyek sorrendje azonban a beolvasás szerinti sorrend marad.

```
#include <stdio.h>

#define N 3

main()
{
  int tomb[ N ], meret = 0, szam;
  scanf( "%d", &szam );
  while ( szam )
  {
    if ( meret < N )
      tomb[ meret++ ] = szam;
    else
    {
      int index = 0, i;
      for ( i = 1; i < N; ++i )
        if ( tomb[ i ] < tomb[ index ] )
          index = i;
      if ( tomb[ index ] < szam )
      {
        while ( index < N - 1 )
        {
          tomb[ index ] = tomb[ index + 1 ];
          ++index;
        }
        tomb[ N - 1 ] = szam;
      }
    }
    scanf( "%d", &szam );
  }
  if ( meret < N )
    printf( "Nem volt %d érték\n", N );
  else
  {
    int i;
    printf( "%d", tomb[ 0 ] );
    for ( i = 1; i < N; ++i )
      printf( ", %d", tomb[ i ] );
    putchar( '\n' );
  }
}
```

1.4. FELADAT. Írjon **programot**, amely a billentyűzetről pozitív egész számokat olvas mindaddig, míg 0-t nem adunk. A program válassza ki a számok közül a **legkisebbeket** és a **legnagyobbakat** (lehetnek azonos értékűek is, de legfeljebb 3-3), írja azok értékét a **képernyőre**, és adja meg, hogy **hányadikként** olvastuk be őket!

```
#include <stdio.h>

main()
{
  int szam, i, legk, legn, min[ 3 ], max[ 3 ], mindb = 0, maxdb;
  for ( i = 1; ; ++i )
  {
    scanf( "%d", &szam );
    if ( szam == 0 )
      break;
    if ( i == 1 )
    {
      legk = legn = szam;
      min[ 0 ] = max[ 0 ] = 1;
      mindb = maxdb = 1;
    }
    else
    {
      if ( szam < legk )
      {
        legk = szam;
        mindb = 1;
        min[ 0 ] = i;
      }
      else if ( szam > legn )
      {
        legn = szam;
        maxdb = 1;
        max[ 0 ] = i;
      }
      else
      {
        if ( szam == legk )
          min[ mindb++ ] = i;
        if ( szam == legn )
          max[ maxdb++ ] = i;
      }
    }
  }
  if ( mindb == 0 )
    puts( "Nem volt egy érték sem." );
  else
  {
    printf( "A legkisebb érték: %d.\nElőfordulásai: ", legk );
    for ( i = 0; i < mindb; ++i )
      printf( "%d ", min[ i ] );
    printf( "\nA legnagyobb érték: %d.\nElőfordulásai: ", legn );
    for ( i = 0; i < maxdb; ++i )
      printf( "%d ", max[ i ] );
    putchar( '\n' );
  }
}
```

1.2. Egydimenziós tömbök

1.5. FELADAT. Írjon **eljárást**, amely paraméterként megkap egy **karaktereket** tartalmazó, **tetszőleges méretű egydimenziós** tömböt, és a tömb **nem betű** karaktereit kicseréli **szóközre**.

```
#include <ctype.h>

void nembetu( char t[], int meret )
{
  int i;
  for ( i = 0; i < meret; ++i )
    if ( !isalpha( t[ i ] ) )
      t[ i ] = ' ';
}
```

1.6. FELADAT. Írjon **eljárást**, amely paraméterként megkap egy **tetszőleges** méretű, **egészeket** tartalmazó **egydimenziós** tömböt. Az eljárás határozza meg a tömbben lévő **pozitív**, **negatív** és **nulla** elemek darabszámát! **Az eljárás nem írhat a képernyőre!**

A feladat többféleképpen is megoldható.

(a) Nézzük először azt a megoldást, amikor az eljárás globális változókban határozza meg a pozitív, negatív és nulla elemek darabszámát:

```
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;
}
```

(b) Egy másik megoldási lehetőség, ha az eljárásnak átadunk még három paramétert, azoknak a memóriaterületeknek a címét, ahol a keresett értékeket tárolni szeretnénk:

```
void poznegnull( int *t, int meret, int *pozitiv, int *negativ, int *nulla )
{
  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;
}
```

1.7. FELADAT. Írjon **eljárást**, amely paraméterként megkap egy **tetszőleges** méretű, **sztringeket** tartalmazó **vektort**, és előállít egy olyan **vektort**, amelymek elemei rendre a paraméterként kapott vektor elemeinek annyiadik **karakterét** tartalmazzák, amennyi az adott elem **indexe**, illetve a `@` karaktert, ha nem létezik ilyen elem. Egy sztring karaktereit 0-tól sorszámozzuk. **Az eljárásban nem lehet semmilyen I/O művelet!**

```
#include <stdlib.h>
#include <string.h>

char *vektor;

void kukacvektor( char *tomb[], int meret )
{
  int i;
  vektor = ( char * )malloc( meret * sizeof( char ) );
  for ( i = 0; i < meret; ++i )
    vektor[ i ] = strlen( tomb[ i ] ) <= i ? '@' : tomb[ i ][ i ];
}
```

1.8. FELADAT. Adva van egy **táblázat**, amelynek kulcsa egész, érték része sztring típusú és maximum 500 elem fér el benne. Írjon **függvényt**, amely **paraméterként** megkapja a táblázatot, az aktuális elemszámot és egy egészet. A függvény **bináris kereséssel** keresse meg az adott egész által meghatározott kulcsú elemet és adja vissza annak **érték részét**! Ha nincs ilyen elem, a függvény az **üres sztringgel** térjen vissza!

```
typedef struct tablaelem
{
  int kulcs;
  char *ertek;
} TABLA[ 500 ];

char *binkeres( TABLA t, int elemszam, int kulcs )
{
  int also = 0, felso = elemszam - 1;
  while ( also <= felso )
  {
    int kozepso = ( also + felso ) / 2;
    if ( t[ kozepso ].kulcs == kulcs )
      return t[ kozepso ].ertek;
    if ( t[ kozepso ].kulcs > kulcs )
      felso = kozepso - 1;
    else
      also = kozepso + 1;
  }
  return "";
}
```

1.9. FELADAT. Írjon egy **programot**, amely a **billentyűzetről** beolvas egy pozitív egész számot (értéke maximum **110** lehet), majd a billentyűzetről beolvas ennyi darab **valós** számot és közülük **képernyőre** írja azokat, amelyek értékének a beolvasott számok **átlagától** való eltérése az átlag felénél nagyobb!

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

main()
{
  double tomb[ 110 ], atlag = 0;
  int i, meret;
  printf( "Méret: " );  scanf( "%d", &meret );
  for ( i = 0; i < meret; ++i )
  {
    scanf( "%lf", &tomb[ i ] );
    atlag += tomb[ i ];
  }
  atlag /= meret;
  for ( i = 0; i < meret; ++i )
```

```
    if ( fabs( tomb[ i ] - atlag ) > atlag / 2 )
      printf( "%lf\n", tomb[ i ] );
}
```

1.10. FELADAT. Írjon **programot**, amely **billentyűzetről** pozitív valós számokat olvas be mindaddig, amíg **nullát** nem adunk (tudjuk, hogy maximum **100** szám lehet). A program írja egy **most létrehozott** szöveges állományba azokat a beolvasott számokat, amelyeknek a **számok átlagától való eltérése** nagyobb, mint az **átlag fele**!

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

#define MERET 100

main()
{
  FILE *fout;
  double tomb[ MERET ], atlag = 0, szam;
  int darab = 0, i;
  fout = fopen( "atlagfel.txt", "w" );
  scanf( "%lf", &szam );
  while ( szam != 0 )
  {
    atlag += tomb[ darab++ ] = szam;
    scanf( "%lf", &szam );
  }
  if ( darab > 0 )
  {
    atlag /= darab;
    for ( i = 0; i < darab; ++i )
      if ( fabs( tomb[ i ] - atlag ) > atlag / 2 )
        fprintf( fout, "%lf\n", tomb[ i ] );
  }
  fclose( fout );
}
```

1.11. FELADAT. Írjon **eljárást**, amely egy paraméterként megkapott, **karaktereket** tartalmazó **egydimenziós** tömbben meghatározza azon rész **kezdő-** és **végindexét**, amelyben **azonos karakterek** vannak! Több ilyen esetén válassza ki azt, amelyben a karakterek száma **maximális**!

```
void azonosresz( char t[], int meret, int *kezdo, int *veg )
{
  int k = 0, v;
  *kezdo = *veg = 0;
  for ( v = 1; v < meret; ++v )
    if ( t[ v ] != t[ v - 1 ] )
      k = v;
    else if ( v - k > *veg - *kezdo )
    {
      *kezdo = k;
      *veg = v;
    }
}
```

1.12. FELADAT. Írjon **programot**, amely billentyűzetről egyenként beolvas egész értékeket addig, amíg a -1 értéket nem kapja. A program írja **képernyőre** a beolvasott számok azon szekvenciáját, amely a **leghosszabb szigorúan monoton csökkenő** sorozatot alkotja!

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

```
main()
{
  int i, *sorozat = NULL, kezdo = 0, veg = -1, maxkezdo = 0, maxveg = 0;
  do
  {
    sorozat = ( int * )realloc( sorozat, ( ++veg + 1 ) * sizeof( int ) );
    scanf( "%d", &sorozat[ veg ] );
    if ( veg != 0 &&
      ( sorozat[ veg ] >= sorozat[ veg - 1 ] || sorozat[ veg ] == -1 ) )
    {
      if ( veg - 1 - kezdo > maxveg - maxkezdo )
      {
        maxkezdo = kezdo;
        maxveg = veg - 1;
      }
      kezdo = veg;
    }
  } while ( sorozat[ veg ] != -1 );
  for ( i = maxkezdo; i <= maxveg && sorozat[ i ] != -1; ++i )
    printf( "%d ", sorozat[ i ] );
  putchar( '\n' );
}
```

1.13. FELADAT. Írjon **eljárást**, amely egy paraméterként kapott **tetszőleges méretű**, egészeket tartalmazó egydimenziós tömbben meghatározza a legnagyobb összegű résztömb **kezdő-** és **végindexét** két output paraméterében!

A feladatot többféleképpen is meg lehet oldani.

(a) Álljon itt először a mezítlábas (brute force) megoldás, amelyben három egymásba ágyazott ciklust futtatunk: egyet a kezdőindexre, egyet a végindexre és egyet a részösszegek kiszámítására.

```
void maxosszresz( int t[], int meret, int *kezdo, int *veg )
{
  int k, v, i;
  long maxosszeg = t[ 0 ];
  *kezdo = *veg = 0;
  for ( k = 0; k < meret; ++k )
    for ( v = k; v < meret; ++v )
    {
      long osszeg = 0;
      for ( i = k; i <= v; ++i )
        osszeg += t[ i ];
      if ( osszeg > maxosszeg )
      {
        maxosszeg = osszeg;
        *kezdo = k;
        *veg = v;
      }
    }
}
```

(b) A második megoldásban a részösszegek kiszámítását nem ciklussal végezzük, hanem egy tömb segítségével. Igy elegendő lesz két egymásba ágyazott for-ciklus az eredmény kiszámításához.

```
void maxosszresz( int t[], int meret, int *kezdo, int *veg )
{
  int k, v, i, *reszosszeg = ( int * )malloc( ( meret + 1 ) * sizeof( int ) );
  long maxosszeg = t[ 0 ];
```

```
  *kezdo = *veg = 0;
  reszosszeg[ 0 ] = 0;
  for ( i = 1; i <= meret; ++i )
    reszosszeg[ i ] = reszosszeg[ i - 1 ] + t[ i - 1 ];
  for ( k = 0; k < meret; ++k )
    for ( v = k; v < meret; ++v )
      if ( reszosszeg[ v + 1 ] - reszosszeg[ k ] > maxosszeg )
      {
        maxosszeg = reszosszeg[ v + 1 ] - reszosszeg[ k ];
        *kezdo = k;
        *veg = v;
      }
  free( reszosszeg );
}
```

(c) Végül lássuk azt a megoldást, amely egyetlen for-ciklussal határozza meg a legnagyobb összegű résztömb kezdő- és végindexét.

```
void maxosszresz( int t[], int meret, int *kezdo, int *veg )
{
  int reszosszeg, maxosszeg, i, k;
  maxosszeg = reszosszeg = *t;
  *kezdo = *veg = k = 0;
  for ( i = 1; i < meret; ++i )
  {
    if ( reszosszeg >= 0 )
      reszosszeg += t[ i ];
    else
    {
      reszosszeg = t[ i ];
      k = i;
    }
    if ( reszosszeg > maxosszeg )
    {
      maxosszeg = reszosszeg;
      *kezdo = k;
      *veg = i;
    }
  }
}
```

1.14. FELADAT. Írjon **eljárást**, amely egy paraméterként megadott, **sztringeket** tartalmazó egydimenziós tömböt **elemeinek hossza** szerint **csökkenő** sorrendbe rendez!

A feladat megoldásához a `qsort()` könyvtári függvényt használjuk. A rendezést követően az azonos hosszúságú sztringek egymáshoz viszonyított elhelyezkedése implementációfüggő.

```
#include <stdlib.h>  /* qsort() */
#include <string.h>

int rendezo( const void *egyik, const void *masik )
{
  return strlen( *( char ** )masik ) - strlen( *( char ** )egyik );
}

void quickcsoksztring( char *t[], int meret )
{
  qsort( t, meret, sizeof( char * ), rendezo );
}
```

Lásd még a 1.15. feladat megoldását!

1.15. FELADAT. Írjon egy **eljárást**, amely a paraméterként megkapott, **tetszőleges méretű**, **sztringeket** tartalmazó, egydimenziós tömböt a sztringek **hosszának csökkenő** sorrendjébe teszi! Azonos hosszak esetén a sztringek sorrendje az eredeti sorrend legyen!

A feladatban leírt egydimenziós tömb rendezésére bármely olyan rendező algoritmus alkalmas, amely nem változtatja meg a tömb azonos hosszúságú elemeinek relatív sorrendjét. Egy ilyen algoritmus a közvetlen beszúró rendezés:

```
#include <string.h>

void csoksztringhossz( char *t[], int meret )
{
  int j;
  for ( j = 1; j < meret; ++j )
  {
    char *kulcs = t[ j ];
    int i = j - 1;
    while ( i >= 0 && strlen( t[ i ] ) < strlen( kulcs ) )
    {
      t[ i + 1 ] = t[ i ];
      --i;
    }
    t[ i + 1 ] = kulcs;
  }
}
```

Lásd még a 1.14. feladat megoldását!

1.16. FELADAT. Írjon **eljárást**, amely paraméterként megkap egy **azonos hosszúságú sztringeket** tartalmazó, **tetszőleges méretű egydimenziós** tömböt, továbbá egy **karaktert** és egy **pozitív egész** számot, és a **képernyőre írja** azokat a tömbelemeket, amelyekben a karakter pontosan az adott számszor fordul elő. **A szélsőséges eseteket vizsgálni kell**!

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

void azonos( char *t[], int meret, char kar, int szam )
{
  int i;
  if ( szam > strlen( t[ 0 ] ) || strlen( t[ 0 ] ) == 0 )
    return;
  for ( i = 0; i < meret; ++i )
  {
    int db = 0;
    char *p;
    for ( p = t[ i ]; *p; ++p )
      if ( *p == kar )
        ++db;
    if ( db == szam )
      puts( t[ i ] );
  }
}
```

1.17. FELADAT. Írjon **eljárást**, amely egy paraméterként megkapott, angol szavakat tartalmazó egydimenziós tömbben meghatározza és **képernyőre** írja a szavak **gyakoriságát**!

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

```
void sztringtombgyak( char *t[], int meret )
{
  int i;
  for ( i = 0; i < meret; ++i )
  {
    int j;
    for ( j = 0; j < i; ++j )
      if ( strcmp( t[ i ], t[ j ] ) == 0 )
        break;
    if ( j == i )  /* ha nem szerepelt korábban */
    {
      int darab = 1;
      for ( j = i + 1; j < meret; ++j )
        if ( strcmp( t[ i ], t[ j ] ) == 0 )
          ++darab;
      printf( "%s: %d\n", t[ i ], darab );
    }
  }
}
```

1.18. FELADAT. Írjon **függvényt**, amely paraméterként megkap egy olyan **tetszőleges** méretű **egydimenziós** tömböt, amely **angol** szavakat tartalmaz. A függvény **visszatérési értéke** adja meg a szavak **gyakoriságát**.

A feladat többféleképpen is megoldható, attól függően, hogy milyen adatszerkezetben, illetve hogyan tároljuk és adjuk vissza a szavak gyakoriságát.

(a) Nézzük először azt a megoldást, amikor egy táblázatot adunk vissza, amelyet egy dinamikusan lefoglalt tárterület kezdőcímével, valamint az ott elhelyezkedő elemek darabszámával reprezentálunk:

```
#include <stdlib.h>
#include <string.h>

typedef struct
{
  char *szo;
  int gyakorisag;
} ELEM;

typedef struct
{
  ELEM *tomb;
  int meret;
} GYAK;

GYAK gyakszo( char *t[], int meret )
{
  GYAK gy = { NULL, 0 };
  int i;
  for ( i = 0; i < meret; ++i )
  {
    int j;
    for ( j = 0; j < gy.meret && strcmp( gy.tomb[ j ].szo, t[ i ] ); ++j )
      ;
    if ( j < gy.meret )
      ++gy.tomb[ j ].gyakorisag;
    else
    {
      gy.tomb = ( ELEM * )realloc( gy.tomb, ( gy.meret + 1 ) * sizeof( ELEM ) );
      gy.tomb[ gy.meret ].szo = t[ i ];
```

```
      gy.tomb[ gy.meret++ ].gyakorisag = 1;
    }
  }
  return gy;
}
```

(b) A következő megoldás egy egyirányban láncolt listában adja meg a keresett adatokat:

```
#include <stdlib.h>
#include <string.h>

typedef struct listaelem
{
  char *szo;
  int gyakorisag;
  struct listaelem *kov;
} LISTAELEM;

LISTAELEM *gyakszo( char *t[], int meret )
{
  LISTAELEM *lista = NULL;
  int i;
  for ( i = 0; i < meret; ++i )
  {
    LISTAELEM *seged = lista, *elozo = NULL;
    while ( seged && strcmp( seged->szo, t[ i ] ) )
    {
      elozo = seged;
      seged = seged->kov;
    }
    if ( seged )
      ++seged->gyakorisag;
    else
    {
      seged = ( LISTAELEM * )malloc( sizeof( LISTAELEM ) );
      seged->szo = t[ i ];
      seged->gyakorisag = 1;
      seged->kov = NULL;
      if ( elozo )
        elozo->kov = seged;
      else
        lista = seged;
    }
  }
  return lista;
}
```

(c) Lássuk végül a keresőfás megoldást:

```
#include <stdlib.h>
#include <string.h>

typedef struct faelem
{
  char *szo;
  int gyakorisag;
  struct faelem *bal, *jobb;
} FAELEM;
```

```
FAELEM *beszur( FAELEM *fa, char *szo )
{
  if ( !fa )
  {
    fa = ( FAELEM * )calloc( 1, sizeof( FAELEM ) );
    fa->szo = szo;
    fa->gyakorisag = 1;
  }
  else if ( strcmp( fa->szo, szo ) > 0 )
    fa->bal = beszur( fa->bal, szo );
  else if ( strcmp( fa->szo, szo ) < 0 )
    fa->jobb = beszur( fa->jobb, szo );
  else
    ++fa->gyakorisag;
  return fa;
}

FAELEM *gyakszo( char *t[], int meret )
{
  FAELEM *fa = NULL;
  int i;
  for ( i = 0; i < meret; ++i )
    fa = beszur( fa, t[ i ] );
  return fa;
}
```

1.19. FELADAT. Írjon **eljárást**, amely egy paraméterként megkapott, **sztringeket** tartalmazó **egydimenziós tömb** minden elemét **azonos hosszúságúra** (a maximális hosszúságú elem hosszúságára) hozza úgy, hogy a sztringek **elejére** megfelelő számú **szóközt** szúr be!

```
#include <stdlib.h>
#include <string.h>

void eloretolt( char *t[], int meret )
{
  int i, maxhossz = 0;
  for ( i = 0; i < meret; ++i )
    if ( strlen( t[ i ] ) > maxhossz )
      maxhossz = strlen( t[ i ] );
  for ( i = 0; i < meret; ++i )
  {
    int j, akthossz = strlen( t[ i ] );
    t[ i ] = realloc( t[ i ], ( maxhossz + 1 ) * sizeof( char ) );
    for ( j = maxhossz; akthossz >= 0; --j, --akthossz )
      t[ i ][ j ] = t[ i ][ akthossz ];
    for ( ; j >= 0; --j )
      t[ i ][ j ] = ' ';
  }
}
```

1.20. FELADAT. Írjon **függvényt**, amely egy egydimenziós, sztringeket tartalmazó tömböt kap **paraméterként**, és annak minden elemét **csonkítja** a legrövidebb elem hosszára, és visszaadja az **új** tömböt!

```
#include <stdlib.h>
#include <string.h>

char **sztringcsonkolo( char *t[], int meret )
{
  char **uj = ( char ** )malloc( meret * sizeof( char * ) );
```

```
  int i, min = strlen( t[ 0 ] );
  for ( i = 1; i < meret; ++i )
    if ( strlen( t[ i ] ) < min )
      min = strlen( t[ i ] );
  for ( i = 0; i < meret; ++i )
  {
    uj[ i ] = ( char * )malloc( ( min + 1 ) * sizeof( char ) );
    strncpy( uj[ i ], t[ i ], min );
    uj[ i ][ min ] = '\0';
  }
  return uj;
}
```

1.21. FELADAT. Írjon **eljárást**, amely egy paraméterként kapott, **sztringeket** tartalmazó **egydimenziós** tömb minden elemét az **elején** és a **végén egyenletesen elosztott** szóközökkel kiegészíti olyan **hosszúságúra**, mint a **leghosszabb** elem hossza! Az eredeti tömb nem módosulhat!

```
#include <stdlib.h>
#include <string.h>

void eloszt( char *t[], int meret, char ***uj )
{
  int maxhossz = 0, i;
  for ( i = 0; i < meret; ++i )
    if ( strlen( t[ i ] ) > maxhossz )
      maxhossz = strlen( t[ i ] );
  *uj = ( char ** )malloc( meret * sizeof( char * ) );
  for ( i = 0; i < meret; ++i )
  {
    int j;
    ( *uj )[ i ] = ( char * )malloc( ( maxhossz + 1 ) * sizeof( char ) );
    strcpy( ( *uj )[ i ], "" );
    for ( j = 0; j < ( maxhossz - strlen( t[ i ] ) ) / 2; ++j )
      strcat( ( *uj )[ i ], " " );
    strcat( ( *uj )[ i ], t[ i ] );
    for ( j = strlen( ( *uj )[ i ] ); j < maxhossz; ++j )
      strcat( ( *uj )[ i ], " " );
  }
}
```

1.3. Sztringek

1.22. FELADAT. Írjon **logikai függvényt**, amely egy paraméterként megkapott sztringnél igaz értéket ad vissza, ha a sztring **tükörszimmetrikus** (pl. görög, kosarasok)!

A feladat a következő megfogalmazásban is szerepelt a beugrókon:

Írjon **logikai függvényt**, amely egy paraméterként megkapott **sztringről** eldönti, hogy az **palindróma-e**!

```
#include <string.h>

int tukorszo( char *s )
{
  int i, j;
  for ( i = 0, j = strlen( s ) - 1; i < j; ++i, --j )
    if ( s[ i ] != s[ j ] )
      return 0;
  return 1;
}
```

1.23. FELADAT. Írjon egy **logikai függvényt**, amely egy paraméterként megkapott **sztring** esetén **igaz** értékkel tér vissza, ha a sztringben a betűk (angol ábécé!) száma **nagyobb**, mint a nem-betű karakterek száma, és **hamissal** tér vissza egyébként!

```
#include <ctype.h>

int tobb_betu( char *s )
{
  int betu = 0;
  while ( *s )
  {
    if ( isalpha( *s ) )
      ++betu;
    else
      --betu;
    s++;
  }
  return betu > 0;
}
```

1.24. FELADAT. Írjon egy **függvényt**, amely egy paraméterként megkapott **sztringben szóközzel** felülírja a **nem betű** karaktereket és visszaadja az új sztringet!

```
#include <ctype.h>   /* isalpha() */
#include <stdlib.h>  /* malloc()  */
#include <string.h>  /* strlen()  */

char *nembetu( char *s )
{
  char *uj = ( char * )malloc( ( strlen( s ) + 1 ) * sizeof( char ) );
  int i;
  for ( i = 0; s[ i ]; ++i )
    uj[ i ] = isalpha( s[ i ] ) ? s[ i ] : ' ';
  uj[ i ] = '\0';
  return uj;
}
```

1.25. FELADAT. Írjon **logikai függvényt**, amely egy paraméterként megkapott **angol** szó esetén igazzal tér vissza, ha a szóban **nincs egymás mellett két mássalhangzó**!

```
#include <ctype.h>
#include <string.h>

int nincsketmsh( char *s )
{
  char *msh = "bcdfghjklmnpqrstvwxyz";
  int i;
  for ( i = 0; i + 1 < strlen( s ); ++i )
    if ( strchr( msh, tolower( s[ i ] ) ) && strchr( msh, tolower( s[ i+1 ] ) ) )
      return 0;
  return 1;
}
```

1.26. FELADAT. Írjon **logikai függvényt**, amely egy paraméterként megkapott sztringnél igaz értéket ad vissza, ha a sztringben van olyan **4 elemű** részsztring, amely **legalább háromszor** ismétlődik.

A feladat többféleképpen is értelmezhető, és többféleképpen is megoldható.

(a) Az első megoldásban karakterenként végezzük az összehasonlítást, és megengedjük a részsztringek közötti átfedést:

```
#include <string.h>

int negyismet( char *s )
{
  int i;
  for ( i = 0; i + 5 < strlen( s ); ++i )
  {
    int j, darab = 1;
    for ( j = i + 1; j + 5 - darab < strlen( s ); ++j )
      if ( s[ i ] == s[ j ] && s[ i+1 ] ==s [ j+1 ] && s[ i+2 ] == s[ j+2 ] &&
           s[ i+3 ] == s[ j+3 ] && ++darab == 3 )
        return 1;
  }
  return 0;
}
```

(b) A második megoldásban a részsztringek összehasonlítását az `strncmp()` könyvtári függvényre bízzuk, és nem engedjük meg a részsztringek között az átfedést:

```
#include <string.h>

int negyismet( char *s )
{
  int i;
  for ( i = 0; i + 11 < strlen( s ); ++i )
  {
    int j, darab = 1;
    for ( j = i + 4; j + 11 - 4 * darab < strlen( s ); ++j )
      if ( !strncmp( s + i, s + j, 4 ) )
      {
        if ( ++darab == 3 )
          return 1;
        j += 3;
      }
  }
  return 0;
}
```

1.27. FELADAT. Írjon logikai **függvényt**, amely egy paraméterként kapott **sztringről** eldönti, hogy van-e benne olyan **részsztring**, amely **pontosan 4 azonos** karakterből áll!

```
#include <string.h>

int pontnegy( char *s )
{
  int i;
  for ( i = 0; i + 3 < strlen( s ); ++i )
    if ( s[ i ] == s[ i+1 ] && s[ i ] == s[ i+2 ] && s[ i ] == s[ i+3 ] )
      return 1;
  return 0;
}
```

1.28. FELADAT. Írjon **függvényt**, amely egy paraméterként megkapott **sztringben** az egymás mellett álló **szóközök** közül csak egyet hagy meg, és visszatér az **új** sztringgel!

```
#include <stdlib.h>
#include <string.h>

char *szokoztelenito( char *s )
```

```
{
  char *uj = ( char * )malloc( ( strlen( s ) + 1 ) * sizeof( char ) );
  int i, j;
  for ( i = 0, j = 0; i < strlen( s ); ++i )
    if ( i == 0 || s[ i ] != ' ' || s[ i-1 ] != ' ' )
      uj[ j++ ] = s[ i ];
  uj[ j ] = '\0';
  return uj;
}
```

1.29. FELADAT. Írjon egy **függvényt**, amelynek első paramétere egy **sztring**, második paramétere egy **pozitív egész szám**, és a függvény adja vissza azt a sztringet, amely az eredetiből úgy keletkezik, hogy azt az elején és a végén kiegészítjük **szóközökkel** (egyenletesen elosztva azokat) úgy, hogy az új sztring hossza a második paraméter értéke legyen!

```
#include <stdlib.h>
#include <string.h>

char *eloszt( char *s, int hossz )
{
  char *uj;
  if ( strlen( s ) >= hossz )
  {
    uj = ( char * )malloc( ( strlen( s ) + 1 ) * sizeof( char ) );
    strcpy( uj, s );
  }
  else
  {
    int i;
    uj = ( char * )malloc( ( hossz + 1 ) * sizeof( char ) );
    for ( i = 0; i < ( hossz - strlen( s ) ) / 2; ++i )
      uj[ i ] = ' ';
    uj[ i ] = '\0';
    strcat( uj, s );
    for ( i = strlen( uj ); i < hossz; ++i )
      uj[ i ] = ' ';
    uj[ i ] = '\0';
  }
  return uj;
}
```

1.30. FELADAT. Írjon **logikai függvényt**, amely akkor tér vissza igaz értékkel, ha a paraméterként kapott **sztring szabványos C egész literál**.

A feladatot többféleképpen is meg lehet oldani.

(a) Első megoldásunkban állapotátmenet-gráfot használunk.

```
#include <ctype.h>

int egeszliteral( char *s )
{
  enum { START, JO, ROSSZ, DECIMALIS, OKT_HEXA, OKTALIS, HEXA_ELSO, HEXA,
         L, U, LU } allapot = START;
  while ( allapot != JO && allapot != ROSSZ )
  {
    switch ( allapot )
    {
      case START:     if ( *s == '0' )
                        allapot = OKT_HEXA;
```

```
                    else if ( isdigit( *s ) )
                      allapot = DECIMALIS;
                    else
                      allapot = ROSSZ;
                    break;
        case DECIMALIS: if ( !*s )
                      allapot = JO;
                    else if ( tolower( *s ) == 'u' )
                      allapot = U;
                    else if ( tolower( *s ) == 'l' )
                      allapot = L;
                    else if ( !isdigit( *s ) )
                      allapot = ROSSZ;
                    break;
        case OKT_HEXA:  if ( !*s )
                      allapot = JO;
                    else if ( tolower( *s ) == 'x' )
                      allapot = HEXA_ELSO;
                    else if ( *s >= '0' && *s <= '7' )
                      allapot = OKTALIS;
                    else if ( tolower( *s ) == 'u' )
                      allapot = U;
                    else if ( tolower( *s ) == 'l' )
                      allapot = L;
                    else
                      allapot = ROSSZ;
                    break;
        case OKTALIS:   if ( !*s )
                      allapot = JO;
                    else if ( tolower( *s ) == 'u' )
                      allapot = U;
                    else if ( tolower( *s ) == 'l' )
                      allapot = L;
                    else if ( !( *s >= '0' && *s <= '7' ) )
                      allapot = ROSSZ;
                    break;
        case HEXA_ELSO: if ( !*s )
                      allapot = JO;
                    else if ( isxdigit( *s ) )
                      allapot = HEXA;
                    else
                      allapot = ROSSZ;
                    break;
        case HEXA:      if ( !*s )
                      allapot = JO;
                    else if ( tolower( *s ) == 'u' )
                      allapot = U;
                    else if ( tolower( *s ) == 'l' )
                      allapot = L;
                    else if ( !isxdigit( *s ) )
                      allapot = ROSSZ;
                    break;
        case L:         if ( !*s )
                      allapot = JO;
                    else if ( tolower( *s ) == 'u' )
                      allapot = LU;
                    else
                      allapot = ROSSZ;
```

```
                    break;
      case U:       if ( !*s )
                      allapot = JO;
                    else if ( tolower( *s ) == 'l' )
                      allapot = LU;
                    else
                      allapot = ROSSZ;
                    break;
      case LU:      if ( !*s )
                      allapot = JO;
                    else
                      allapot = ROSSZ;
                    break;
    }
    ++s;
  }
  return allapot == JO;
}
```

(b) A második megoldásban egyszerű sztringkezelést használunk.

```
#include <ctype.h>

#define FALSE 0
#define TRUE  !FALSE

int egeszliteral( char *s )
{
  int i;
  if ( !isdigit( s[ 0 ] ) )
    return FALSE;
  if ( s[ 0 ] == '0' )
    if ( tolower( s[ 1 ] ) == 'x' )
    {
      if ( !isxdigit( s[ 2 ] ) )
        return FALSE;
      for ( i = 3; isxdigit( s[ i ] ); ++i )
        ;
    }
    else
      for ( i = 1; s[ i ] >= '0' && s[ i ] <= '7'; ++i )
        ;
  else
    for ( i = 0; isdigit( s[ i ] ); ++i )
      ;
  if ( !s[ i ] )
    return TRUE;
  if ( tolower( s[ i ] ) == 'u' )
    return !s[ i+1 ] || tolower( s[ i+1 ] ) == 'l' && !s[ i+2 ];
  if ( tolower( s[ i ] ) == 'l' )
    return !s[ i+1 ] || tolower( s[ i+1 ] ) == 'u' && !s[ i+2 ];
  return FALSE;
}
```

1.4. Kétdimenziós tömbök

1.31. FELADAT. Írjon **logikai függvényt**, amely egy paraméterként megkapott, **sztringeket** tartalmazó négyzetes mátrixról eldönti, hogy **szimmetrikus-e**!

```
#include <string.h>

int negyzetes( char *s[], int meret )
{
  int i, j;
  for ( i = 0; i < meret - 1; ++i )
    for ( j = i + 1; j < meret; ++j )
      if ( strcmp( s[ i * meret + j ], s[ j * meret + i ] ) )
        return 0;
  return 1;
}
```

1.32. FELADAT. Írjon **eljárást**, amely paraméterként megkapott, **tetszőleges méretű**, egészeket tartalmazó kvadratikus mátrixot **tükröz** a **mellékátlójára**!

```
void tukroz( int *t, int meret )
{
  int i, j;
  for ( i = 0; i < meret - 1; ++i )
    for ( j = 0; j < meret - 1 - i; ++j )
    {
      int seged = t[ i * meret + j ];
      t[ i * meret + j ] = t[ ( meret - j - 1 ) * meret + ( meret - i - 1 ) ];
      t[ ( meret - j - 1 ) * meret + ( meret - i - 1 ) ] = seged;
    }
}
```

1.33. FELADAT. Írjon **eljárást**, amely paraméterként megkap egy **bitmátrixot** és egy **bitvektort**, majd a bitmátrix minden oszlopa és a bitvektor között **kizáró vagy** műveletet végez! Az eredeti bitmátrixra a továbbiakban nincs szükség.

```
void xor( int *m, int *v, int sor, int oszlop )
{
  int i, j;
  for ( j = 0; j < oszlop; ++j )
    for ( i = 0; i < sor; ++i )
      m[ i * oszlop + j ] ^= v[ i ];
}
```

1.34. FELADAT. Írjon **függvényt**, amely **tetszőleges** méretű, **valós** értékeket tartalmazó **kétdimenziós** tömböt kap paraméterként. A függvény határozza meg azon **oszlop indexét**, amelyben van olyan elem, amelynek az értéke megegyezik az oszlop elemeinek **átlagával**! Ha több ilyen oszlop is van, akkor a **legnagyobb** indexértéket adja vissza!

```
int atlagindex( double *t, int sor, int oszlop )
{
  int j;
  for ( j = oszlop - 1; j >= 0; --j )
  {
    double atlag = 0;
    int i;
    for ( i = 0; i < sor; ++i )
      atlag += t[ i * oszlop + j ];
    atlag /= sor;
    for ( i = 0; i < sor; ++i )
      if ( t[ i * oszlop + j ] == atlag )
        return j;
  }
  return -1;
}
```

Megjegyezzük, hogy a `t[ i * oszlop + j ] == atlag` kifejezés értéke az oszlopátlag lebegőpontos ábrázolásának pontatlansága miatt igen gyakran akkor is hamis, ha matematikailag megegyezik vele a tömbelem értéke. Ezért a következő megoldást javasoljuk:

```
#include <math.h>   /* fabs() */
#include <float.h>  /* DBL_EPSILON */

int atlagindex( double *t, int sor, int oszlop )
{
  int j;
  for ( j = oszlop - 1; j >= 0; --j )
  {
    double atlag = 0;
    int i;
    for ( i = 0; i < sor; ++i )
      atlag += t[ i * oszlop + j ];
    atlag /= sor;
    for ( i = 0; i < sor; ++i )
      if ( fabs( t[ i * oszlop + j ] - atlag ) < DBL_EPSILON )
        return j;
  }
  return -1;
}
```

1.35. FELADAT. Írjon **függvényt**, amely egy paraméterként kapott, **egészeket** tartalmazó **kétdimenziós** tömb azon oszlopának **indexét** adja vissza, amelyben a **legkevesebb pozitív elem** van!

Amennyiben több oszlopban is annyi pozitív elem van, mint abban, amelyikben a legkevesebb pozitív elem található, akkor az alábbi függvény a legelső ilyen oszlop indexét határozza meg.

```
int kevespozoszlop( int *t, int sor, int oszlop )
{
  int j, min = sor, minoszlop = 0;
  for ( j = 0; j < oszlop; ++j )
  {
    int i, poz = 0;
    for ( i = 0; i < sor; ++i )
      if ( t[ i * oszlop + j ] > 0 )
        ++poz;
    if ( poz < min )
    {
      min = poz;
      minoszlop = j;
    }
  }
  return minoszlop;
}
```

1.36. FELADAT. Írjon egy **függvényt**, amely egy paraméterként megkapott, **egészeket** tartalmazó **kétdimenziós** tömb 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;
}
```

1.37. FELADAT. Írjon **eljárást**, amely paraméterként megkap egy **tetszőleges** méretű, **valósakat** tartalmazó **kétdimenziós** tömböt, és előállít egy olyan **egydimenziós** tömböt, amely a **sorok átlagát** tartalmazza. **Az eljárás a képernyőre nem írhat!**

```
#include <stdlib.h>

double *soratl;

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

1.38. FELADAT. Írjon **eljárást**, amely egy paraméterként megkapott, **egészeket** tartalmazó **kétdimenziós tömb** oszlopait úgy rendezi át, hogy az első sor elemei nagyság szerint **csökkenő sorrendben** legyenek! Feltehetjük, hogy az első sor elemei különbözőek.

A feladatot többféle módon is meg lehet oldani, itt buborékrendezéssel rendeztük a tömb első sorának elemeit:

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

1.39. FELADAT. Írjon **eljárást**, amely egy paraméterként megkapott, tetszőleges méretű, valósakat tartalmazó kétdimenziós tömb **sorait** úgy rendezi át, hogy az **utolsó oszlop** értékei **csökkenő** sorrendben legyenek!

A feladatot többféle módon is meg lehet oldani, itt maximumkiválasztásos rendezéssel rendeztük a tömb utolsó oszlopának elemeit:

```
void atrendez( double *t, int sor, int oszlop )
{
  int i;
  for ( i = 0; i < sor - 1; ++i )
  {
    int j, k, index = i;
    for ( k = i + 1; k < sor; ++k )
      if ( t[ index * oszlop + oszlop - 1 ] < t[ k * oszlop + oszlop - 1 ] )
        index = k;
    for ( j = 0; j < oszlop; ++j )  /* a sorok minden elemét cseréljük */
    {
      double seged = t[ index * oszlop + j ];
      t[ index * oszlop + j ] = t[ i * oszlop + j ];
      t[ i * oszlop + j ] = seged;
    }
  }
}
```

1.40. FELADAT. Írjon **eljárást**, amely egy paraméterként megkapott, tetszőleges méretű, egészeket tartalmazó kétdimenziós tömb **oszlopátlagai** közül meghatározza a **legnagyobbat** (több ilyen is lehet)! **Az eljárás nem írhat képernyőre és állományba!**

Vegyük észre, hogy a feladat megfogalmazása csalafinta: hiába van esetleg több azonos legnagyobb oszlopátlag, az eljárásnak csak ezek egyikét, egyetlen értéket kell meghatároznia.

A feladat többféleképpen is megoldható.

(a) Lássuk először azt a megoldást, amikor az eljárás egy globális változóban határozza meg a keresett (legnagyobb) oszlopátlagot:

```
double atlag;

void atlagol( int *t, int sor, int oszlop )
{
  int i, j;
  atlag = 0.0;
  for ( i = 0; i < sor; ++i )
    atlag += t[ i * oszlop ];
  for ( j = 1; j < oszlop; ++j )
  {
    double seged = 0.0;
    for ( i = 0; i < sor; ++i )
      seged += t[ i * oszlop + j ];
    if ( seged > atlag )
      atlag = seged;
  }
  atlag /= sor;
}
```

(b) Egy másik megoldási lehetőség, ha az eljárásnak átadunk még egy paramétert, annak a memóriaterületnek a címét, ahol a keresett átlagértéket tárolni szeretnénk:

```
void atlagol( int *t, int sor, int oszlop, double *atlag )
{
  int i, j;
  *atlag = 0.0;
  for ( i = 0; i < sor; ++i )
```

```
        *atlag += t[ i * oszlop ];
      for ( j = 1; j < oszlop; ++j )
      {
        double seged = 0.0;
        for ( i = 0; i < sor; ++i )
          seged += t[ i * oszlop + j ];
        if ( seged > *atlag )
          *atlag = seged;
      }
      *atlag /= sor;
    }
```

1.41. FELADAT. Írjon **függvényt**, amely paraméterként megkap egy **tetszőleges méretű**, **egészeket** tartalmazó **kvadratikus mátrixot**, és visszaadja a **főátló** maximális és minimális elemét. **A képernyőre nem írunk!**

```
typedef struct
{
  int max, min;
} MAXMIN;

MAXMIN foatlo( int *m, int meret )
{
  MAXMIN mm = { *m, *m };
  int i;
  for ( i = 1; i < meret; ++i )
    if ( m[ i * ( meret + 1 ) ] > mm.max )
      mm.max = m[ i * ( meret + 1 ) ];
    else if ( m[ i * ( meret + 1 ) ] < mm.min )
      mm.min = m[ i * ( meret + 1 ) ];
  return mm;
}
```

1.42. FELADAT. Írjon **függvényt**, amely paraméterként megkap egy **tetszőleges méretű**, **valósakat** tartalmazó **kétdimenziós** tömböt, és visszatérési értékként meghatározza a **sorok átlagának minimumát** és az **oszlopok átlagának maximumát**.

```
typedef struct { double min, max; } MINMAX;

MINMAX sorminoszmax( double *m, int sor, int oszlop )
{
  MINMAX mm;
  int i, j;
  for ( i = 0; i < sor; ++i )
  {
    double osszeg = 0;
    for ( j = 0; j < oszlop; ++j )
      osszeg += m[ i * oszlop + j ];
    if ( i == 0 || osszeg < mm.min )
      mm.min = osszeg;
  }
  mm.min /= oszlop;
  for ( j = 0; j < oszlop; ++j )
  {
    double osszeg = 0;
    for ( i = 0; i <sor; ++i )
      osszeg += m[ i * oszlop + j ];
    if ( j == 0 || osszeg > mm.max )
      mm.max = osszeg;
```

```
  }
  mm.max /= sor;
  return mm;
}
```

1.43. FELADAT. Írjon **eljárást**, amely egy paraméterként megkapott, **egészeket** tartalmazó **kétdimenziós** tömbben meghatározza azon oszlopok **indexét** (akárhány ilyen lehet), amelyekben a negatív elemek száma **legalább kétszerese** a nulla értékű elemek számának! A tömb mérete **tetszőleges**.

A feladat többféleképpen is megoldható.

(a) Először is lássuk azt a megoldást, amely két globális változót használ: egyik a feltételnek megfelelő oszlopok darabszámát fogja tartalmazni, a másik pedig arra a memóriaterületre mutat, ahol a keresett oszlopindexeket tároljuk.

```
#include <stdlib.h>

int *dupneg;
int darab;

void negketszer( int *t, int sor, int oszlop )
{
  int j;
  dupneg = NULL;
  darab = 0;
  for ( j = 0; j < oszlop; ++j )
  {
    int i, negativ = 0, nulla = 0;
    for ( i = 0; i < sor; ++i )
      if ( t[ i * oszlop + j ] < 0 )
        ++negativ;
      else if ( t[ i * oszlop + j ] == 0 )
        ++nulla;
    if ( negativ >= 2 * nulla )
    {
      dupneg = ( int * )realloc( dupneg, darab + 1 );
      dupneg[ darab++ ] = j;
    }
  }
}
```

(b) Az eljárás természetesen paraméterlistán keresztül is kommunikálhat a hívó programegységgel. Ekkor a megoldás a következő lehet:

```
#include <stdlib.h>

void negketszer( int *t, int sor, int oszlop, int *darab, int **dupneg )
{
  int j;
  *dupneg = NULL;
  *darab = 0;
  for ( j = 0; j < oszlop; ++j )
  {
    int i, negativ = 0, nulla = 0;
    for ( i = 0; i < sor; ++i )
      if ( t[ i * oszlop + j ] < 0 )
        ++negativ;
      else if ( t[ i * oszlop + j ] == 0 )
        ++nulla;
    if ( negativ >= 2 * nulla )
```

```
      {
        *dupneg = ( int * )realloc( *dupneg, *darab + 1 );
        ( *dupneg )[ ( *darab )++ ] = j;
      }
    }
  }
```

(c) Álljon itt végül az a megoldást, amely egy globális mutatóval dolgozik: a mutató egy olyan memóriaterületre mutat, amelynek első eleme az ezt követő elemek (a keresett oszlopindexek) darabszámát adja meg.

```
#include <stdlib.h>

int *dupneg;

void negketszer( int *t, int sor, int oszlop )
{
  int j;
  dupneg = ( int * )malloc( sizeof( int ) );
  *dupneg = 0;
  for ( j = 0; j < oszlop; ++j )
  {
    int i, negativ = 0, nulla = 0;
    for ( i = 0; i < sor; ++i )
      if ( t[ i * oszlop + j ] < 0 )
        ++negativ;
      else if ( t[ i * oszlop + j ] == 0 )
        ++nulla;
    if ( negativ >= 2 * nulla )
    {
      dupneg = ( int * )realloc( dupneg, *dupneg + 2 );
      dupneg[ ++*dupneg ] = j;
    }
  }
}
```

1.44. FELADAT. Írjon **eljárást**, amely egy **paraméterként** megadott kétdimenziós, egészeket tartalmazó tömb azon **oszlopát** határozza meg, amelyben benne van az egész tömb **legnagyobb** eleme (csak egy ilyen van)!

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

void legoszlop( int *t, int sor, int oszlop, int **oszl )
{
  int i, j, maxelem = *t, maxoszlop = 0;
  *oszl = ( int * )malloc( sor * sizeof( int ) );
  for ( i = 0; i < sor; ++i )
    for ( j = 0; j < oszlop; ++j )
      if ( t[ i * oszlop + j ] > maxelem )
      {
        maxelem = t[ i * oszlop + j ];
        maxoszlop = j;
      }
  for ( i = 0; i < sor; ++i )
    ( *oszl )[ i ] = t[ i * oszlop + maxoszlop ];
}
```

1.45. FELADAT. Írjon egy **eljárást**, amely egy paraméterként megkapott, **tetszőleges méretű**, egészeket tartalmazó kvadratikus mátrix **főátlójában** elhelyezi **soronként** a főátló fölötti elemek **összegét**!

```
#include <stdio.h>

void atlossz( int *t, int meret )
{
  int j;
  for ( j = 0; j < meret; ++j )
  {
    int i;
    t[ j * meret + j ] = 0;
    for ( i = 0; i < j; ++i )
      t[ j * meret + j ] += t[ i * meret + j ];
  }
}
```

1.46. FELADAT. Írjon **eljárást**, amely egy paraméterként megkapott, **egészeket** tartalmazó, **tetszőleges méretű kétdimenziós** tömb minden olyan elemének a **0** értéket adja, amelynek a **tömb elemeinek átlagától való eltérése az átlag felénél nagyobb**! Az eredeti tömböt változatlanul kell hagyni.

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

void nullaz( int *t, int sor, int oszlop, int **ujtomb )
{
  *ujtomb = ( int * )malloc( sor * oszlop * sizeof( int ) );
  int i, j;
  double atlag;
  for ( i = 0; i < sor; ++i )
    for ( j = 0; j < oszlop; ++j )
      atlag += t[ i * oszlop + j ];
  atlag /= sor * oszlop;
  for ( i = 0; i < sor; ++i )
    for ( j = 0; j < oszlop; ++j )
      if ( fabs( t[ i * oszlop + j ] - atlag ) > atlag / 2 )
        ( *ujtomb )[ i * oszlop + j ] = 0;
      else
        ( *ujtomb )[ i * oszlop + j ] = t[ i * oszlop + j ];
}
```

1.5. Fák

1.47. FELADAT. Írjon **logikai függvényt**, amely egy paraméterként megkapott, **egészeket** tartalmazó, **szigorú értelemben vett bináris fáról** eldönti, hogy **tökéletesen kiegyensúlyozott-e**!

```
typedef struct faelem
{
  int adat;
  struct faelem *bal, *jobb;
} FAELEM;

int elemszam( FAELEM *fa )
{
  return fa ? elemszam( fa->bal ) + elemszam( fa->jobb ) + 1 : 0;
}

int tokkiegy( FAELEM *fa )
{
  return !fa || tokkiegy( fa->bal ) && tokkiegy( fa->jobb) &&
                abs( elemszam( fa->bal ) - elemszam( fa->jobb ) ) <= 1;
}
```

1.48. FELADAT. Írjon egy **eljárást**, amely egy paraméterként megkapott, **karaktereket** tartalmazó **bináris fában** a kisbetűket nagybetűre cseréli!

```
#include <ctype.h>

typedef struct faelem
{
  char kar;
  struct faelem *bal, *jobb;
} FAELEM;

void nagybetu( FAELEM *fa )
{
  if ( fa )
  {
    fa->kar = toupper( fa->kar );
    nagybetu( fa->bal );
    nagybetu( fa->jobb );
  }
}
```

1.49. FELADAT. Adva van egy **tetszőleges egészeket** tartalmazó **bináris** fa. Írjon **függvényt**, amely paraméterként megkapja a bináris fa **gyökérmutatóját**, a fában elhelyezett értékekből felépít egy **keresőfát**, és visszaadja annak **gyökérmutatóját**! Az új fa elemeinek szerkezete: **kulcs** (a különböző egész értékek), **gyakoriság** (az eredeti fában az adott kulcs hányszor fordult elő).

```
#include <stdlib.h>

typedef struct binfa {
  int ertek;
  struct binfa *bal, *jobb;
} BINFA;

typedef struct keresofa {
  int kulcs;
  int gyakorisag;
  struct keresofa *bal, *jobb;
} KERESOFA;

KERESOFA *keresofa_bovit( KERESOFA *k, int ertek )
{
  if ( !k )
  {
    k = ( KERESOFA * )calloc( 1, sizeof( KERESOFA ) );
    k->kulcs = ertek;
    k->gyakorisag = 1;
  }
  else if ( k->kulcs > ertek )
    k->bal = keresofa_bovit( k->bal, ertek );
  else if ( k->kulcs < ertek )
    k->jobb = keresofa_bovit( k->jobb, ertek );
  else
    k->gyakorisag++;
  return k;
}

KERESOFA *binfa_inorder( KERESOFA *k, BINFA *b )
{
  if ( b )
```

```
  {
    k = binfa_inorder( k, b->bal );
    k = keresofa_bovit( k, b->ertek );
    k = binfa_inorder( k, b->jobb );
  }
  return k;
}

KERESOFA *keresofa_epito( BINFA *b )
{
  return binfa_inorder( NULL, b );
}
```

1.50. FELADAT. Írjon **eljárást**, amely egy paraméterként megkapott, **egészeket** tartalmazó **keresőfából** kitöröl egy szintén paraméterként megadott **értéket**! Az eljárás írjon megfelelő hibaüzenetet a képernyőre, ha a törlés valamilyen okból nem hajtható végre!

Egy elem törlésénél a következő lehetőségek fordulhatnak elő:

- A törlendő elem nincs benne a keresőfában. Ekkor hibaüzenetet kell a képernyőre írni.
- A törlendő elem levélelem, azaz nincs egyetlen rákövetkezője sem.
- A törlendő elemnek csak egy rákövetkezője van.
- A törlendő elemnek két rákövetkezője van.

A fentieket figyelembe véve a feladat rekurzívan és iteratívan is megoldható. Az alábbiakban három megoldást adunk meg.

(a) Az első megoldás rekurzívan oldja meg a feladatot. Egy külön rekurzív eljárásban kezeltük benne azt az esetet, amikor a törlendő elemnek két rákövetkezője van.

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

typedef struct faelem {
  int adat;
  struct faelem *bal, *jobb;
} FAELEM;

void torol_2rakov( FAELEM *torlendo, FAELEM **legjobb )
{
  if( ( *legjobb )->jobb )
    torol_2rakov( torlendo, &( *legjobb )->jobb );
  else
  {
    FAELEM *seged = *legjobb;
    torlendo->adat = ( *legjobb )->adat;
    *legjobb = ( *legjobb )->bal;
    free( seged );
  }
}

void torol( FAELEM **gym, int ertek )
{
  if( !*gym )
    fputs( "Nincs ilyen érték a fában!\n", stderr );
  else if( ( *gym )->adat != ertek )
    torol( ( *gym )->adat < ertek ? &( *gym )->jobb : &( *gym )->bal, ertek );
  else if( !( *gym )->bal || !( *gym )->jobb )
```

```
  {
    FAELEM *torlendo = *gym;
    *gym = ( *gym )->bal ? ( *gym )->bal : ( *gym )->jobb;
    free( torlendo );
  }
  else
    torol_2rakov( *gym, &( *gym )->bal );
}
```

(b) Másodikként lássuk az iteratív megoldást:

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

typedef struct faelem {
  int adat;
  struct faelem *bal, *jobb;
} FAELEM;

void torol( FAELEM **gym, int ertek )
{
  FAELEM *akt = *gym, *szulo = NULL;
  while ( akt != NULL && akt->adat != ertek )
  {
    szulo = akt;
    akt = akt->adat > ertek ? akt->bal : akt->jobb;
  }
  if ( !akt )
    fputs( "Nincs ilyen érték a fában!\n", stderr );
  else if ( !akt->bal || !akt->jobb )
  {
    if( !szulo )
      *gym = akt->bal ? akt->bal : akt->jobb;
    else if ( akt->adat < szulo->adat )
      szulo->bal = akt->bal ? akt->bal : akt->jobb;
    else
      szulo->jobb = akt->bal ? akt->bal : akt->jobb;
    free( akt );
  }
  else
  {
    FAELEM *seged = akt->jobb;
    szulo = akt;
    while ( seged->bal )
    {
      szulo = seged;
      seged = seged->bal;
    }
    if ( szulo != akt )
      szulo->bal = seged->jobb;
    else
      szulo->jobb = seged->jobb;
    akt->adat = seged->adat;
    free( seged );
  }
}
```

(c) Végül álljon itt egy olyan rekurzív megoldás, amely iteratív elemeket is tartalmaz (azoknál az elemeknél, amelyeknek két rákövetkezőjük is van):

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

typedef struct faelem {
  int adat;
  struct faelem *bal, *jobb;
} FAELEM;

void torol( FAELEM **gym, int ertek )
{
  if( !*gym )
    fputs( "Nincs ilyen érték a fában!\n", stderr );
  else if( ( *gym )->adat != ertek )
    torol( ( *gym )->adat < ertek ? &( *gym )->jobb : &( *gym )->bal, ertek );
  else if( !( *gym )->bal || !( *gym )->jobb )
  {
    FAELEM *torlendo = *gym;
    *gym = ( *gym )->bal ? ( *gym )->bal : ( *gym )->jobb;
    free( torlendo );
  }
  else
  {
    FAELEM *seged = ( *gym )->jobb;
    while ( seged->bal )
      seged = seged->bal;
    ( *gym )->adat = seged->adat;
    torol( &( *gym )->jobb, seged->adat );
  }
}
```

1.6. Kifejezések

1.51. FELADAT. Írjon **függvényt**, amely paraméterként egy olyan sztringet kap, amely egy szabályos, **teljesen zárójelezett infix** kifejezést tartalmaz, és meghatározza a kifejezés fájának **magasságát**! A kifejezésben csak a $+$, $-$, $*$, / bináris operátorok és maximum 3 jegyű egész szám operandusok fordulnak elő.

```
int infixmagas( char *s )
{
  int magas = 1, max = 0;
  while ( *s )
  {
    if ( *s == '(' )
      ++magas;
    else if ( *s == ')' )
      --magas;
    if ( magas > max )
      max = magas;
    s++;
  }
  return max;
}
```

1.52. FELADAT. Írjon **programot**, amely **billentyűzetről** beolvas egy **szabályos, teljesen zárójelezett C kifejezést**, amely operandusként csak **konstansokat** és **változókat** tartalmaz, és a **képernyőre** írja azt a **részkifejezést**, amelyet **először** kell kiértékelni!

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

```
main()
{
  char kif[ 200 ], *p;
  printf( "Kérem a kifejezést: " );  gets( kif );
  for ( p = kif; *p && *p != ')'; ++p )
    ;
  if ( *p )
  {
    while ( *--p != '(' )
      ;
    while ( *++p != ')' )
      putchar( *p );
    putchar( '\n' );
  }
  else
    puts( kif );
}
```

1.53. FELADAT. Írjon **programot**, amely billentyűzetről megkap egy szabályos **prefix** alakú kifejezést. A program írja **képernyőre** az **elsőnek** kiértékelentő részkifejezést **infix** alakban! A kifejezés csak a +, −, ∗, / kétoperandusú operátorokat és operandusként olyan változókat tartalmaz, amelyek neve egyetlen karakterből áll.

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define FALSE 0
#define TRUE  !FALSE

main()
{
  char *op = "+-*/", kif[ 3 ];
  int megvan = FALSE, ch;
  while ( ( ch = getchar() ) != EOF )
    if ( !megvan && !isspace( ch ) )
    {
      kif[ 0 ] = kif[ 1 ];
      kif[ 1 ] = kif[ 2 ];
      kif[ 2 ] = ch;
      if ( !strchr( op, kif[ 1 ] ) && !strchr( op, kif[ 2 ] ) )
        megvan = TRUE;
    }
  if ( megvan )
    printf( "%c%c%c\n", kif[ 1 ], kif[ 0 ], kif[ 2 ] );
  else
    printf( "%c\n", kif[ 2 ] );
}
```

1.54. FELADAT. Adott egy csak a +, -, /, * **bináris** operátorokat tartalmazó **szabályos** kifejezés fájának **postorder** bejárásával kapott sorozat. Az operátorokat és az operandusokat **egy szóköz** választja el egymástól. A sorozatot billentyűzetről kapjuk. Írjon **programot**, amely képernyőre írja a kifejezés **prefix** alakját!

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

typedef struct faelem
{
```

```
  char *op;
  struct faelem *bal, *jobb;
} FAELEM;

typedef struct veremelem
{
  FAELEM *elem;
  struct veremelem *kov;
} VEREMELEM;

VEREMELEM *verem;   /* NULL értékkel inicializálódik */

void push( FAELEM *elem )
{
  VEREMELEM *ujelem = ( VEREMELEM * ) malloc( sizeof( VEREMELEM ) );
  ujelem->elem = elem;
  ujelem->kov = verem;
  verem = ujelem;
}

FAELEM *pop()
{
  FAELEM *seged = verem->elem;
  VEREMELEM *torlendo = verem;
  verem = verem->kov;
  free( torlendo );
  return seged;
}

void preorder( FAELEM *gyoker )
{
  if ( gyoker )
  {
    printf( "%s ", gyoker->op );
    free( gyoker->op );
    preorder( gyoker->bal );
    preorder( gyoker->jobb );
    free( gyoker );
  }
}

main()
{
  char kif[ 256 ];
  puts( "Kérem a postfix kifejezést:" );
  while ( scanf( "%s", kif ) != EOF )
  {
    FAELEM *uj = ( FAELEM * )malloc( sizeof( FAELEM ) );
    uj->op = ( char * )malloc( ( strlen( kif ) + 1 ) * sizeof( char ) );
    strcpy( uj->op, kif );
    if ( strlen( kif ) == 1 && strchr( "+-*/", *kif ) )
    {
      uj->jobb = pop();
      uj->bal = pop();
    }
    else
      uj->bal = uj->jobb = NULL;
    push( uj );
```

```
  }
  preorder( pop() );
  putchar( '\n' );
}
```

1.55. FELADAT. Írjon **programot**, amely **billentyűzetről** beolvas egy olyan **teljesen zárójelezett kifejezést**, mely csak a $+$, $-$, $*$, / operátorokat és olyan **változó** operandusokat tartalmaz, amelyek nevében csak **betű** szerepel! Tudjuk, hogy a kifejezésben **zárójelhiba** van. A program írja képernyőre a kifejezést, és jelölje meg a hiba helyét!

A feladat többféleképpen is megoldható.

(a) Az első megoldásban karakterenként végighaladunk a kifejezésen, és minden karakterről eldöntjük, hogy állhat-e az adott helyen. A szóközöket átugorva, a vizsgált és az azt megelőző karakter alapján állítjuk be a `hiba` változó értékét, ha kell.

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>

typedef enum
  { OK, OPERANDUS, OPERATOR, NYITO, ZARO, ERVENYTELEN, SOKZARO, KEVESZARO }
  HIBAOK;
char operatorok[] = "+-*/";

main()
{
  int zarojel = 0, operandus[ 1000 ] = { 0 }, i = 0;
  HIBAOK hiba = OK;
  char kif[ 1000 ], elozo = '\0';;
  printf( "A kifejezés: " );
  fgets( kif, 1000, stdin );
  while ( kif[ i ] != '\n' && kif[ i ] != '\0' )
  {
    while ( kif[ i ] == ' ' || kif[ i ] == '\t' )
      ++i;
    if ( isalpha( kif[ i ] ) )
    {
      if ( isalpha( elozo ) || elozo == ')' )
      {
        hiba = OPERANDUS;
        break;
      }
      elozo = kif[ i ];
      while ( isalpha( kif[ i ] ) )
        ++i;
      ++operandus[ zarojel ];
    }
    else if ( strchr( operatorok, kif[ i ] ) )
    {
      if ( strchr( operatorok, elozo ) || elozo == '(' ||
           operandus[ zarojel ] == 2 || zarojel == 0 )
      {
        hiba = OPERATOR;
        break;
      }
      elozo = kif[ i++ ];
    }
    else if ( kif[ i ] == '(' )
    {
```

```
    if ( isalpha( elozo ) || elozo == ')' )
    {
      hiba = NYITO;
      break;
    }
    operandus[ ++zarojel ] = 0;
    elozo = kif[ i++ ];
  }
  else if ( kif[ i ] == ')' )
  {
    if ( strchr( operatorok, elozo ) || elozo == '(' )
    {
      hiba = ZARO;
      break;
    }
    if ( --zarojel < 0 )
    {
      hiba = SOKZARO;
      break;
    }
    ++operandus[ zarojel ];
    elozo = kif[ i++ ];
  }
  else
  {
    hiba = ERVENYTELEN;
    break;
  }
}
puts( kif );
if ( hiba == OK && zarojel > 0 )
  hiba = KEVESZARO;
if ( hiba == OK )
  puts( "Nincs hiba a kifejezésben." );
else
{
  int j;
  for ( j = 0; j < i; ++j )
    putchar( ' ' );
  printf( "^\n" );
  switch ( hiba )
  {
    case OPERANDUS:
      puts( "Itt nem állhat operandus!" );
      break;
    case OPERATOR:
      puts( "Itt nem állhat operátor!" );
      break;
    case NYITO:
      puts( "Itt nem állhat nyitó zárójel!" );
      break;
    case ZARO:
      puts( "Itt nem állhat záró zárójel!" );
      break;
    case ERVENYTELEN:
      puts( "Érvénytelen karakter!" );
      break;
    case SOKZARO:
```

```
        puts( "Túl sok a záró zárójel!" );
        break;
      case KEVESZARO:
        puts( "Hiányzó záró zárójel!" );
        break;
    }
  }
}
```

(b) A következő megoldásban állapotátmenet-gráfot használtunk a kifejezés feldolgozására.

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define HIBA 100
#define VEGE 101

enum { KEZDO, OPERATOR, OPERANDUS, NYITO, ZARO, PARATLANZARO, ERVENYTELEN,
       BETU, NYITOVEG, OPERANDUSVEG, OPERATORVEG, KETOPERATOR, HIANYZOZARO,
       SOKOPERATOR };

main()
{
  char kif[ 1000 ];
  int i, allapot = 1, muvjel, *verem = NULL, darab = 0, hibakod = -1;

  printf( "A kifejezés: " );  fgets( kif, 1000, stdin );

  if ( kif[ strlen( kif ) - 1 ] == '\n' )
    kif[ strlen( kif ) - 1 ] = '\0';

  i = 0;
  while ( allapot != VEGE )
  {
    char ch = kif[ i ];
    switch ( allapot )
    {
      case 1:    if ( ch == '\0' )
                   allapot = VEGE;
                 else if ( isspace( ch ) )
                   putchar( ch );
                 else if ( isalpha( ch ) || ch == '_' )
                 {
                   putchar( ch );
                   allapot = 2;
                 }
                 else if ( ch == '(' )
                 {
                   putchar( ch );
                   muvjel = 0;
                   allapot = 4;
                 }
                 else
                 {
                   printf( "<hiba>%c", ch );
                   hibakod = KEZDO;
                   allapot = HIBA;
```

```
            }
            break;
case 2:     if ( ch == '\0' )
              allapot = VEGE;
            else if ( isalpha( ch ) || ch == '_' )
              putchar( ch );
            else if ( isspace( ch ) )
            {
              putchar( ch );
              allapot = 3;
            }
            else
            {
              printf( "<hiba>%c", ch );
              if ( strchr( "+-*/", ch ) )
                hibakod = OPERATOR;
              else if ( ch == '(' )
                hibakod = NYITO;
              else if ( ch == ')' )
                hibakod = ZARO;
              else
                hibakod = ERVENYTELEN;
              allapot = HIBA;
            }
            break;
case 3:     if ( ch == '\0' )
              allapot = VEGE;
            else if ( isspace( ch ) )
              putchar( ch );
            else
            {
              printf( "<hiba>%c", ch );
              if ( strchr( "+-*/", ch ) )
                hibakod = OPERATOR;
              else if ( ch == '(' )
                hibakod = NYITO;
              else if ( ch == ')' )
                hibakod = PARATLANZARO;
              else if ( isalpha( ch ) || ch == '_' )
                hibakod = BETU;
              else
                hibakod = ERVENYTELEN;
              allapot = HIBA;
            }
            break;
case 4:     if ( ch == '\0' )
            {
              printf( "<hiba>" );
              hibakod = NYITOVEG;
              allapot = VEGE;
            }
            else if ( isspace( ch ) )
              putchar( ch );
            else if ( ch == '(' )
            {
              putchar( ch );
              ++darab;
              verem = ( int * )realloc( verem, darab * sizeof( int ) );
```

```
                verem[ darab - 1 ] = muvjel;
              }
              else if ( isalpha( ch ) || ch == '_' )
              {
                putchar( ch );
                allapot = 5;
              }
              else
              {
                printf( "<hiba>%c", ch );
                if ( ch == ')' )
                  hibakod = ZARO;
                else if ( strchr( "+-*/", ch ) )
                  hibakod = OPERATOR;
                else
                  hibakod = ERVENYTELEN;
                allapot = HIBA;
              }
              break;
    case 5:   if ( ch == '\0' )
              {
                printf( "<hiba>" );
                hibakod = OPERANDUSVEG;
                allapot = VEGE;
              }
              else if ( isalpha( ch ) || ch == '_' )
                putchar( ch );
              else if ( isspace( ch ) )
              {
                putchar( ch );
                allapot = 6;
              }
              else if ( strchr( "+-*/", ch ) != NULL )
              {
                putchar( ch );
                muvjel = 1;
                allapot = 7;
              }
              else
              {
                printf( "<hiba>%c", ch );
                if ( ch == '(' )
                  hibakod = NYITO;
                else if ( ch == ')' )
                  hibakod = ZARO;
                else
                  hibakod = ERVENYTELEN;
                allapot = HIBA;
              }
              break;
    case 6:   if ( ch == '\0' )
              {
                printf( "<hiba>" );
                hibakod = OPERANDUSVEG;
                allapot = VEGE;
              }
              else if ( isspace( ch ) )
                putchar( ch );
```

```
                else if ( strchr( "+-*/", ch ) != NULL )
                {
                  putchar( ch );
                  muvjel = 1;
                  allapot = 7;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  if ( isalpha( ch ) || ch == '_' )
                    hibakod = OPERANDUS;
                  else if ( ch == '(' )
                    hibakod = NYITO;
                  else if ( ch == ')' )
                    hibakod = ZARO;
                  else
                    hibakod = ERVENYTELEN;
                  allapot = HIBA;
                }
                break;
        case 7: if ( ch == '\0' )
                {
                  printf( "<hiba>" );
                  hibakod = OPERATORVEG;
                  allapot = VEGE;
                }
                else if ( ch == '(' )
                {
                  putchar( ch );
                  ++darab;
                  verem = ( int * )realloc( verem, darab * sizeof( int ) );
                  verem[ darab - 1 ] = muvjel;
                  allapot = 4;
                }
                else if ( isspace( ch ) )
                  putchar( ch );
                else if ( isalpha( ch ) || ch == '_' )
                {
                  putchar( ch );
                  allapot = 8;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  if ( ch == ')' )
                    hibakod = ZARO;
                  else if ( strchr( "+-*/", ch ) )
                    hibakod = KETOPERATOR;
                  else
                    hibakod = ERVENYTELEN;
                  allapot = HIBA;
                }
                break;
        case 8: if ( ch == '\0' )
                {
                  printf( "<hiba>" );
                  hibakod = HIANYZOZARO;
                  allapot = VEGE;
```

```
                }
                else if ( isalpha( ch ) || ch == '_' )
                  putchar( ch );
                else if ( isspace( ch ) )
                {
                  putchar( ch );
                  allapot = 9;
                }
                else if ( ch == ')' )
                {
                  putchar( ch );
                  if ( verem != NULL )
                  {
                    muvjel = verem[ darab - 1 ];
                    --darab;
                    verem = ( int * )realloc( verem, darab * sizeof( int ) );
                    allapot = muvjel ? 9 : 6;
                  }
                  else
                    allapot = 3;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  if ( strchr( "+-*/", ch ) )
                    hibakod = SOKOPERATOR;
                  else if ( ch == '(' )
                    hibakod = NYITO;
                  else
                    hibakod = ERVENYTELEN;
                  allapot = HIBA;
                }
                break;
        case 9: if ( ch == '\0' )
                {
                  printf( "<hiba>" );
                  hibakod = HIANYZOZARO;
                  allapot = VEGE;
                }
                else if ( isspace( ch ) )
                  putchar( ch );
                else if ( ch == ')' )
                {
                  putchar( ch );
                  if ( verem != NULL )
                  {
                    muvjel = verem[ darab - 1 ];
                    --darab;
                    verem = ( int * )realloc( verem, darab * sizeof( int ) );
                    if ( muvjel == 0 )
                       allapot = 6;
                  }
                  else
                    allapot = 3;
                }
                else
                {
                  printf( "<hiba>%c", ch );
```

```
                    if ( strchr( "+-*/", ch ) )
                      hibakod = SOKOPERATOR;
                    else if ( ch == '(' )
                      hibakod = NYITO;
                    else if ( isalpha( ch ) || ch == '_' )
                      hibakod = OPERANDUS;
                    else
                      hibakod = ERVENYTELEN;
                    allapot = HIBA;
                  }
                  break;
      case HIBA: if ( ch == '\0' )
                    allapot = VEGE;
                  else
                    putchar( ch );
                  break;
    }
    ++i;
  }
  putchar( '\n' );

  switch ( hibakod )
  {
    case KEZDO:        puts( "Szabálytalan kezdőkarakter." );
                       break;
    case OPERATOR:     puts( "Itt nem állhat operátor." );
                       break;
    case OPERANDUS:    puts( "Itt nem állhat operandus." );
                       break;
    case NYITO:        puts( "Itt nem állhat nyitó zárójel." );
                       break;
    case ZARO:         puts( "Itt nem állhat záró zárójel." );
                       break;
    case PARATLANZARO: puts( "Nincs nyitó párja ennek a záró zárójelnek." );
                       break;
    case ERVENYTELEN:  puts( "Érvénytelen karakter." );
                       break;
    case BETU:         puts( "Itt nem állhat betű." );
                       break;
    case NYITOVEG:     puts( "A kifejezés nem végződhet nyitó zárójellel." );
                       break;
    case OPERANDUSVEG: puts( "Félbehagyott kifejezés, hiányzó operátor." );
                       break;
    case OPERATORVEG:  puts( "Félbehagyott kifejezés, hiányzó operandus." );
                       break;
    case KETOPERATOR:  puts( "Két operátor nem állhat egymás mellett." );
                       break;
    case HIANYZOZARO:  puts( "Hiányzó záró zárójel." );
                       break;
    case SOKOPERATOR:  puts( "Itt nem állhat újabb operátor." );
                       break;
    default:           puts( "Szabályos." );
                       break;
  }

  free( verem );
}
```

1.56. FELADAT. Írjon **programot**, amely **billentyűzetről** megkap egy olyan **teljesen zárójelezett** kifejezést, amely csak a − és a + **egy-** és **kétoperandusú** operátorokat, operandusként pedig olyan **C-beli változókat** tartalmaz, melyek neve **maximum két** karakterből áll. Ellenőrizze le, hogy a kifejezés **szabályos-e**. A képernyőre írjon értelemszerű hibaüzeneteket.

A feladat többféleképpen is megoldható.

(a) A következő program egy környezetfüggetlen generatív grammatika segítségével eldönti, hogy a kifejezés szabályos-e vagy sem. Az algoritmus meglehetősen lassú (elsősorban hosszú kifejezések esetén).

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MERET 1000

char minta[ MERET + 1 ];
int szabalyos;

void keres( char *s )
{
  int i;
  if ( !strcmp( s, minta ) )
    szabalyos = 1;
  for ( i = 0; i < strlen( s ); ++i )
  {
    if ( s[ i ] == 'S' )
    {
      char uj[ MERET + 1 ];
      /* S -> b szabály */
      strcpy( uj, s );
      uj[ i ] = 'b';
      keres( uj );
      /* S -> S_ szabály */
      if ( strlen( s ) < strlen( minta ) )
      {
        strncpy( uj, s, i + 1 );
        uj[ i + 1 ] = '\0';
        strcat( uj, " " );
        strcat( uj, s + i + 1 );
        keres( uj );
      }
      /* S -> _S szabály */
      if ( strlen( s ) < strlen( minta ) )
      {
        strncpy( uj, s, i );
        uj[ i ] = '\0';
        strcat( uj, " " );
        strcat( uj, s + i );
        keres( uj );
      }
      /* S -> bA szabály */
      if ( strlen( s ) < strlen( minta ) )
      {
        strncpy( uj, s, i );
        uj[ i ] = '\0';
        strcat( uj, "bA" );
```

```
        strcat( uj, s + i + 1 );
        keres( uj );
      }
      /* S -> (MS) szabály */
      if ( strlen( s ) + 2 < strlen( minta ) )
      {
        strncpy( uj, s, i );
        uj[ i ] = '\0';
        strcat( uj, "(MS)" );
        strcat( uj, s + i + 1 );
        keres( uj );
      }
      /* S -> (SmS) szabály */
      if ( strlen( s ) + 3 < strlen( minta ) )
      {
        strncpy( uj, s, i );
        uj[ i ] = '\0';
        strcat( uj, "(SmS)" );
        strcat( uj, s + i + 1 );
        keres( uj );
      }
    }
    else if ( s[ i ] == 'A' )
    {
      char uj[ MERET + 1 ];
      /* A -> b szabály */
      strcpy( uj, s );
      uj[ i ] = 'b';
      keres( uj );
      /* A -> s szabály */
      strcpy( uj, s );
      uj[ i ] = 's';
      keres( uj );
    }
    else if ( s[ i ] == 'M' )
    {
      char uj[ MERET + 1 ];
      /* M -> _M szabály */
      if ( strlen( s ) < strlen( minta ) )
      {
        strncpy( uj, s, i );
        uj[ i ] = '\0';
        strcat( uj, " " );
        strcat( uj, s + i );
        keres( uj );
      }
      /* M -> m szabály */
      strcpy( uj, s );
      uj[ i ] = 'm';
      keres( uj );
    }
  }
}

main()
{
  char kif[ MERET + 1 ], szo[ MERET + 1 ];
  int i;
```

```
  printf( "A kifejezés: " );  gets( kif );

  for ( i = 0; i < strlen( kif ); ++i )
    if ( kif[ i ] == '(' || kif[ i ] == ')' )
      minta[ i ] = kif[ i ];
    else if ( isspace( kif[ i ] ) )
      minta[ i ] = ' ';
    else if ( isalpha( kif[ i ] ) || kif[ i ] == '_' )
      minta[ i ] = 'b';
    else if ( isdigit( kif[ i ] ) )
      minta[ i ] = 's';
    else if ( kif[ i ] == '+' || kif[ i ] == '-' )
      minta[ i ] = 'm';
    else
      minta[ i ] = 'x';
  minta[ i ] = '\0';

  strcpy( szo, "S" );

  szabalyos = 0;
  keres( szo );
  printf( "kifejezés: *%s*\n", kif );
  printf( "minta:     *%s*\n", minta );
  if ( szabalyos )
    puts( "A kifejezés szabályos." );
  else
    puts( "A kifejezés nem szabályos." );
}
```

(b) A második megoldásban a levezetési szabályok Chomsky-féle normálalakúak, így alkalmazhatjuk a Cocke–Younger–Kasami-féle algoritmust annak eldöntésére, hogy a begépelt kifejezés eleme-e a grammatika által generált nyelvnek.

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

const char *szabaly[] =
  { "SSK", "SKS", "SBA", "SNE", "EKE", "EMF", "FSZ", "SNG", "GSE" };
const int meret = sizeof szabaly / sizeof( char * );

char **tomb, n;

void inicializal( char *s )
{
  int i;
  n = strlen( s );
  tomb = ( char ** )malloc( n * ( n + 1 ) * sizeof( char * ) / 2 );
  for ( i = 0; i < n * ( n + 1 ) / 2; ++i )
    tomb[ i ] = ( char * )calloc( 1, sizeof( char ) );
}

char *get( int i, int j )
{
  return tomb[ n * ( n+1 ) / 2 - ( n-i-1 ) * ( n-i ) / 2 - ( n-i-j ) ];
}
```

```
void set( int i, int j, char ch )
{
  char *p = tomb[ n * ( n+1 ) / 2 - ( n-i-1 ) * ( n-i ) / 2 - ( n-i-j ) ];
  int hossz = strlen( p );
  p = ( char * )realloc( p, hossz + 2 );
  p[ hossz ] = ch;
  p[ hossz + 1 ] = '\0';
}

void felszabadit()
{
  int i;
  for ( i = 0; i < n * ( n + 1 ) / 2; ++i )
    free( tomb[ i ] );
  free( tomb );
}

main()
{
  char kif[ 1000 ];
  int i, j, min = 0;

  printf( "A kifejezés: " );  gets( kif );

  inicializal( kif );

  for ( i = 0; i < n; ++i )
  {
    if ( isspace( kif[ i ] ) )
      set( n - 1 - i, i, 'K' );                    /*  K -> space       */
    else if ( kif[ i ] == '(' )
      set( n - 1 - i, i, 'N' );                    /*  N -> (           */
    else if ( kif[ i ] == ')' )
      set( n - 1 - i, i , 'Z' );                   /*  Z -> )           */
    else if ( strchr( "+-", kif[ i ] ) )
      set( n - 1 - i, i, 'M' );                    /*  M -> {+,-}       */
    else if ( isdigit( kif[ i ] ) )
      set( n - 1 - i, i, 'A' );                    /*  A -> {számjegy}  */
    else if ( isalpha( kif[ i ] ) || kif[ i ] == '_' )
    {
      set( n - 1 - i, i, 'A' );                    /*  A -> {_,betű}    */
      set( n - 1 - i, i, 'B' );                    /*  B -> {_,betű}    */
      set( n - 1 - i, i, 'S' );                    /*  S -> {_,betű}    */
    }
  }

  for ( i = n - 2; i >= 0; --i )
    for ( j = n - 2 - i; j >= 0; --j )
    {
      int k;
      for ( k = 1; k < n - i - j; ++k )
      {
        int m;
        for ( m = 0; m < meret; ++m )
          if ( strchr( get( i + k, j ), szabaly[ m ][ 1 ] ) &&
               strchr( get( i, n - i - k ), szabaly[ m ][ 2 ] ) &&
               !strchr( get( i, j ), szabaly[ m ][ 0 ] ) )
            set( i, j, szabaly[ m ][ 0 ] );
```

```
      }
    }
  for ( i = 0; i < n; ++i )
    for ( j = 0; j < n - i; ++j )
      if ( strlen( get( i, j ) ) > min )
        min = strlen( get( i, j ) );
  printf( "A kifejezés%s szabályos.\n", n && strchr( *tomb,'S' ) ? "" : " nem" );

  felszabadit();
}
```

(c) Állapotátmenet-gráfot használva hibás kifejezés esetén a hiba okát is meg tudjuk határozni.

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define VEGE 100
#define HIBA 101

enum { KEZDO, OPERATOR, OPERANDUS, NYITO, ZARO, PARATLANZARO, ERVENYTELEN,
       BETU, NYITOVEG, SZAMJEGY, OPERANDUSVEG, OPERATORVEG, KETOPERATOR,
       HIANYZOZARO, SOKOPERATOR, NEMSZAM };

main()
{
  char kif[ 1000 ];
  int i, allapot = 1, muvjel, *verem = NULL, darab = 0, hibakod = -1;

  printf( "A kifejezés: " );  gets( kif );

  i = 0;
  while ( allapot != VEGE )
  {
    char ch = kif[ i ];
    switch ( allapot )
    {
      case 1:    if ( ch == '\0' )
                   allapot = VEGE;
                 else if ( isalpha( ch ) || ch == '_' )
                 {
                   putchar( ch );
                   allapot = 2;
                 }
                 else if ( isspace( ch ) )
                   putchar( ch );
                 else if ( ch == '(' )
                 {
                   putchar( ch );
                   muvjel = 0;
                   allapot = 4;
                 }
                 else
                 {
                   printf( "<hiba>%c", ch );
                   allapot = HIBA;
                   hibakod = KEZDO;
                 }
```

```
                break;
        case 2: if ( ch == '\0' )
                  allapot = VEGE;
                else if ( isspace( ch ) || isalnum( ch ) || ch == '_')
                {
                  putchar( ch );
                  allapot = 3;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  switch ( ch )
                  {
                    case '+': case '-': hibakod = OPERATOR;    break;
                    case '(':           hibakod = NYITO;       break;
                    case ')':           hibakod = ZARO;        break;
                    default:            hibakod = ERVENYTELEN; break;
                  }
                  allapot = HIBA;
                }
                break;
        case 3: if ( ch == '\0' )
                  allapot = VEGE;
                else if ( isspace( ch ) )
                  putchar( ch );
                else
                {
                  printf( "<hiba>%c", ch );
                  switch ( ch )
                  {
                    case '+': case '-': hibakod = OPERATOR;     break;
                    case '(':           hibakod = NYITO;        break;
                    case ')':           hibakod = PARATLANZARO; break;
                    default:
                      if ( isalnum( ch ) || ch == '_' )
                        hibakod = BETU;
                      else
                        hibakod = ERVENYTELEN;
                      break;
                  }
                  allapot = HIBA;
                }
                break;
        case 4: if ( ch == '\0' )
                {
                  printf( "<hiba>" );
                  hibakod = NYITOVEG;
                  allapot = VEGE;
                }
                else if ( isspace( ch ) )
                  putchar( ch );
                else if ( ch == '(' )
                {
                  putchar( ch );
                  ++darab;
                  verem = ( int * )realloc( verem, darab * sizeof( int ) );
                  verem[ darab - 1 ] = muvjel;
                }
```

```
                else if ( isalpha( ch ) || ch == '_' )
                {
                  putchar( ch );
                  allapot = 5;
                }
                else if ( strchr( "+-", ch ) != NULL )
                {
                  putchar( ch );
                  muvjel = 1;
                  allapot = 7;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  if ( ch == ')' )
                    hibakod = ZARO;
                  else if ( isdigit( ch ) )
                    hibakod = SZAMJEGY;
                  else
                    hibakod = ERVENYTELEN;
                  allapot = HIBA;
                }
                break;
        case 5: if ( ch == '\0' )
                {
                  printf( "<hiba>" );
                  hibakod = OPERANDUSVEG;
                  allapot = VEGE;
                }
                else if ( isspace( ch ) || isalnum( ch ) || ch == '_')
                {
                  putchar( ch );
                  allapot = 6;
                }
                else if ( strchr( "+-", ch ) != NULL )
                {
                  putchar( ch );
                  muvjel = 1;
                  allapot = 7;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  switch ( ch )
                  {
                    case '(':           hibakod = NYITO;       break;
                    case ')':           hibakod = ZARO;        break;
                    default:            hibakod = ERVENYTELEN; break;
                  }
                  allapot = HIBA;
                }
                break;
        case 6: if ( ch == '\0' )
                {
                  printf( "<hiba>" );
                  hibakod = OPERANDUSVEG;
                  allapot = VEGE;
                }
```

```
                else if ( isspace( ch ) )
                  putchar( ch );
                else if ( strchr( "+-", ch ) != NULL )
                {
                  putchar( ch );
                  muvjel = 1;
                  allapot = 7;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  switch ( ch )
                  {
                    case '(':           hibakod = NYITO;       break;
                    case ')':           hibakod = ZARO;        break;
                    default:
                      if ( isalpha( ch ) || ch == '_' )
                        hibakod = OPERANDUS;
                      else if ( isdigit( ch ) )
                        hibakod = NEMSZAM;
                      else
                        hibakod = ERVENYTELEN;
                      break;
                  }
                  allapot = HIBA;
                }
                break;
        case 7: if ( ch == '\0' )
                {
                  printf( "<hiba>" );
                  hibakod = OPERATORVEG;
                  allapot = VEGE;
                }
                else if ( isspace( ch ) )
                  putchar( ch );
                else if ( ch == '(' )
                {
                  putchar( ch );
                  ++darab;
                  verem = ( int * )realloc( verem, darab * sizeof( int ) );
                  verem[ darab - 1 ] = muvjel;
                  allapot = 4;
                }
                else if ( isalpha( ch ) || ch == '_' )
                {
                  putchar( ch );
                  allapot = 8;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  if ( ch == ')' )
                    hibakod = ZARO;
                  else if ( isdigit( ch ) )
                    hibakod = SZAMJEGY;
                  else if ( strchr( "+-", ch ) )
                    hibakod = KETOPERATOR;
                  else
```

```
                    hibakod = ERVENYTELEN;
                  allapot = HIBA;
                }
                break;
        case 8:   if ( ch == '\0' )
                {
                  printf( "<hiba>" );
                  hibakod = HIANYZOZARO;
                  allapot = VEGE;
                }
                else if ( isspace( ch ) || isalnum( ch ) || ch == '_' )
                {
                  putchar( ch );
                  allapot = 9;
                }
                else if ( ch == ')' )
                {
                  putchar( ch );
                  if ( darab > 0 )  /* a verem nem üres */
                  {
                    muvjel = verem[ --darab ];
                    verem = realloc( verem, darab * sizeof( int ) );
                    allapot = muvjel ? 9 : 6;
                  }
                  else
                    allapot = 3;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  switch ( ch )
                  {
                    case '+': case '-': hibakod = SOKOPERATOR; break;
                    case '(':           hibakod = NYITO;       break;
                    default:            hibakod = ERVENYTELEN; break;
                  }
                  allapot = HIBA;
                }
                break;
        case 9:   if ( ch == '\0' )
                {
                  printf( "<hiba>" );
                  hibakod = HIANYZOZARO;
                  allapot = VEGE;
                }
                else if ( isspace( ch ) )
                  putchar( ch );
                else if ( ch == ')' )
                {
                  putchar( ch );
                  if ( darab > 0 )  /* a verem nem üres */
                  {
                    muvjel = verem[ --darab ];
                    verem = realloc( verem, darab * sizeof( int ) );
                    if ( !muvjel )
                      allapot = 6;
                  }
                  else
```

```
                    allapot = 3;
                }
                else
                {
                  printf( "<hiba>%c", ch );
                  switch ( ch )
                  {
                    case '(':           hibakod = NYITO;       break;
                    case '+': case '-': hibakod = SOKOPERATOR; break;
                    default:
                      if ( isalpha( ch ) || ch == '_' )
                        hibakod = OPERANDUS;
                      else if ( isdigit( ch ) )
                        hibakod = NEMSZAM;
                      else
                        hibakod = ERVENYTELEN;
                      break;
                  }
                  allapot = HIBA;
                }
                break;
    case HIBA: if ( ch == '\0' )
                  allapot = VEGE;
                else
                  putchar( ch );
                break;
  }
  ++i;
}
putchar( '\n' );

switch ( hibakod )
{
  case KEZDO:        puts( "Szabálytalan kezdőkarakter." );
                     break;
  case OPERATOR:     puts( "Itt nem állhat operátor." );
                     break;
  case OPERANDUS:    puts( "Itt nem állhat operandus." );
                     break;
  case NYITO:        puts( "Itt nem állhat nyitó zárójel." );
                     break;
  case ZARO:         puts( "Itt nem állhat záró zárójel." );
                     break;
  case PARATLANZARO: puts( "Nincs nyitó párja ennek a záró zárójelnek." );
                     break;
  case ERVENYTELEN:  puts( "Érvénytelen karakter." );
                     break;
  case BETU:         puts( "Itt nem állhat betű vagy számjegy." );
                     break;
  case NYITOVEG:     puts( "A kifejezés nem végződhet nyitó zárójellel." );
                     break;
  case SZAMJEGY:     puts( "A változónév nem kezdődhet számjeggyel." );
                     break;
  case OPERANDUSVEG: puts( "Félbehagyott kifejezés, hiányzó operátor." );
                     break;
  case OPERATORVEG:  puts( "Félbehagyott kifejezés, hiányzó operandus." );
                     break;
  case KETOPERATOR:  puts( "Két operátor nem állhat egymás mellett." );
```

```
                        break;
        case HIANYZOZARO:  puts( "Hiányzó záró zárójel." );
                        break;
        case SOKOPERATOR:  puts( "Itt nem állhat újabb operátor." );
                        break;
        case NEMSZAM:      puts( "Itt nem állhat számjegy." );
                        break;
        default:           puts( "Szabályos." );
                        break;
      }
      free( verem );
    }
```

1.7. Állományok

1.57. FELADAT. Írjon **programot**, amely **angol** szavakat kér be **billentyűzetről** *** végjelig, és kiírja egy **szöveges** állományba közülük azokat, amelyek tartalmazzák a `b`, `c`, `x`, `y` karaktereket!

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

main()
{
  FILE *f = fopen( "ki.txt", "w" );
  char szo[ 100 ];
  scanf( "%s", szo );
  while ( strcmp( szo, "***" ) )
  {
    if ( strchr( szo, 'b' ) && strchr( szo, 'c' ) &&
         strchr( szo, 'x' ) && strchr( szo, 'y' ) )
      fprintf( f, "%s\n", szo );
    scanf( "%s", szo );
  }
  fclose( f );
}
```

1.58. FELADAT. Írjon **programot**, amely a **billentyűzetről angol szavakat** olvas mindaddig, amíg **üres sztringet** nem kap. A program írja egy **szöveges állományba** azokat a szavakat, amelyekben egymás mellett van **legalább három mássalhangzó**.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define angmsh( c ) ( strchr( "bcdfghjklmnpqrstvwxyz", tolower( c ) ) )

main()
{
  FILE *f = fopen( "szavak.txt", "w" );
  char szo[ 100 ];
  for ( ; ; )
  {
    char szo[ 100 ];
    int i;
    gets( szo );
    if ( szo[ 0 ] == '\0' )
      break;
    for ( i = 0; i + 2 < strlen( szo ); ++i )
      if ( angmsh( szo[ i ] ) &&  angmsh( szo[ i + 1 ] ) && angmsh( szo[ i + 2 ] ) )
```

```
      {
        fprintf( f, "%s\n", szo );
        break;
      }
  }
  fclose( f );
}
```

1.59. FELADAT. Adva van egy **szöveges állomány**, amely egymástól egy szóközzel elválasztott **különböző angol** szavakat tartalmaz. Írjon **programot**, amely képernyőre írja azokat a szavakat (ezekből akármennyi lehet), amelyekben a **legtöbb magánhangzó** van.

```
#include <ctype.h>
#include <stdio.h>
#include <string.h>

int mghdarab( char *s )
{
  int darab = 0;
  while ( *s )
  {
    if ( strchr( "aeiou", tolower( *s ) ) != NULL )
      ++darab;
    s++;
  }
  return darab;
}

int main()
{
  FILE *fin;
  char input[ 256 ], szo[ 100 ];
  int max = 0;
  printf( "Az input állomány: " );  scanf( "%s", input );
  fin = fopen( input, "r" );
  while ( fscanf( fin, "%s", szo ) != EOF )
  {
    int mgh = mghdarab( szo );
    if ( mgh > max )
      max = mgh;
  }
  fclose( fin );
  fin = fopen( input, "r" );
  while ( fscanf( fin, "%s", szo ) != EOF )
  {
    int mgh = mghdarab( szo );
    if ( mgh == max )
      printf( "%s\n", szo );
  }
  fclose( fin );
}
```

1.60. FELADAT. Adott egy **szöveges** állomány, amelyben **magyar** szavak vannak, minden szó után egy **szóköz** áll. Írjon **eljárást**, amely képernyőre írja azon sorokat (több ilyen is lehet), amelyekben a **legkevesebb szó** van!

Feltételezve, hogy a feldolgozandó állomány neve `be.txt`, és az állomány egyetlen sora sem tartalmaz 2000-nél több karaktert, egy lehetséges megoldás a következő:

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

```
void kevesszosorok()
{
  FILE *f = fopen( "be.txt", "r" );
  char sor[ 2000 ];
  int i, min = 2000;
  while ( fgets( sor, 2000, f ) )
  {
    int szoszam = 0;
    for ( i = 0; sor[ i ]; ++i )
      if ( sor[ i ] == ' ' )
        ++szoszam;
    if ( szoszam < min )
      min = szoszam;
  }
  f = freopen( "be.txt", "r", f );
  while ( fgets( sor, 2000, f ) )
  {
    int szoszam = 0;
    for ( i = 0; sor[ i ]; ++i )
      if ( sor[ i ] == ' ' )
        ++szoszam;
    if ( szoszam == min )
      printf( sor );
  }
  fclose( f );
}
```

1.61. FELADAT. Adva van egy **szöveges állomány**, amely soraiban egymástól **egyetlen szóközzel** elválasztott **magyar** szavak állnak. Írjon **eljárást**, amely meghatározza az állományban előforduló szavak **gyakoriságát**! Feltételezhetjük, hogy maximum 200 különböző szó fordul elő.

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

typedef struct tablaelem
{
  char *kulcs;
  int  gyakorisag;
} TABLA[ 200 ];

int darab;
TABLA tomb;

void gyakszamolo( char *allomany )
{
  FILE *f;
  char szo[ 30 ];
  darab = 0;
  f = fopen( allomany, "r" );
  while ( fscanf( f, "%s", szo ) != EOF )
  {
    int i;
    for ( i = 0; i < darab; ++i )
      if ( strcmp( tomb[ i ].kulcs, szo ) == 0 )
        break;
    if ( i == darab )
    {
```

```
      ++darab;
      tomb[ i ].kulcs = ( char * )malloc( ( strlen( szo ) + 1 ) * sizeof( char ) );
      strcpy( tomb[ i ].kulcs, szo );
      tomb[ i ].gyakorisag = 1;
    }
    else
      ++tomb[ i ].gyakorisag;
  }
  fclose( f );
}
```

1.62. FELADAT. Adva van egy olyan **szöveges állomány**, amely sorai **egyetlen szóközzel** elválasztott angol szavakat tartalmaznak. Írjon **programot**, amely meghatározza és **képernyőre** írja a szövegben előforduló szavak **gyakoriságát**!

Vegyük észre, hogy a feladat nagyon hasonlít a 1.61. feladatban megfogalmazottakhoz, mindössze annyi a különbség, hogy most nem ismerjük a szöveges állományban található, egymástól különböző szavak maximális darabszámát. Ezért a szavakat és a gyakoriságukat tartalmazó táblázatot dinamikusan célszerű létrehozni.

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

typedef struct tablaelem
{
  char *kulcs;
  int  gyakorisag;
} TABLA;

main()
{
  FILE *fin;
  char input[ 256 ], szo[ 30 ];
  TABLA *tabla = NULL;
  int darab = 0, i;

  printf( "Az input állomány: " );  scanf( "%s", input );

  fin = fopen( input, "r" );
  while ( fscanf( fin, "%s", szo ) != EOF )
  {
    for ( i = 0; i < darab; ++i )
      if ( strcmp( tabla[ i ].kulcs, szo ) == 0 )
        break;
    if ( i == darab )  /* még nem szerepelt a szó a táblázatban */
    {
      ++darab;
      tabla = ( TABLA * )realloc( tabla, darab * sizeof( TABLA ) );
      tabla[ i ].kulcs = ( char * )malloc( ( strlen( szo ) + 1 ) * sizeof( char ) );
      strcpy( tabla[ i ].kulcs, szo );
      tabla[ i ].gyakorisag = 1;
    }
    else               /* a szó benne volt a táblázatban */
      ++tabla[ i ].gyakorisag;
  }
  fclose( fin );

  for ( i = 0; i < darab; ++i )
    printf( "%s: %d\n", tabla[ i ].kulcs, tabla[ i ].gyakorisag );
```

```
  free( tabla );
}
```

1.63. FELADAT. Írjon **eljárást**, amely paraméterként megkap **két szöveges állomány nevet** és **egy sztringet**, majd az első állomány azon sorait, melyeknek a **vége** azonos a sztringgel, átírja a másik állományba! Az első állomány létezik, a másodikat most kell létrehozni.

A feladatot többféleképpen is meg lehet oldani. Azonban minden megoldáshoz érdemes felhasználni a következő függvényt, amely meghatározza, hogy egy (létező) állománynak milyen hosszú a leghosszabb sora:

```
#include <stdio.h>

int sorhossz( char *allomany )
{
  int hossz = 0, maxhossz = 0, ch;
  FILE *f = fopen( allomany, "r" );

  while ( ( ch = fgetc( f ) ) != EOF )
  {
    if ( ch == '\n' )
    {
      if ( hossz > maxhossz )
        maxhossz = hossz;
      hossz = 0;
    }
    else
      ++hossz;
  }

  if ( hossz > maxhossz )
    maxhossz = hossz;

  fclose( f );

  return maxhossz;
}
```

A fenti függvény segítségével pontosan akkora memóriaterületet foglalhatunk le egy állománybeli sor beolvasásához, amennyire ahhoz maximálisan szükség lehet.

(a) Ezek után lássuk azt a megoldást, amely az állományból beolvasott sorokat karakterenként hátulról előrefelé haladva dolgozza fel:

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

void sorvegir( char *innev, char *outnev, char *s )
{
  int hossz = sorhossz( innev );
  char *sor = ( char * )malloc( ( hossz + 2 ) * sizeof( char ) );
  FILE *in = fopen( innev, "r" ), *out = fopen( outnev, "w" );
  while ( fgets( sor, hossz + 2, in ) )
  {
    char *sorp = sor + strlen( sor ) - 1, *sp = s + strlen( s ) - 1;
    if ( *sorp == '\n' )
      --sorp;
    while ( sorp >= sor && sp >= s && *sorp == *sp )
    {
```

```
        --sorp;
        --sp;
      }
      if ( sp < s )
        fputs( sor, out );
    }
    fclose( in );
    fclose( out );
    free( sor );
  }
```

(b) Most pedig következzék az a megoldás, amely a sorokat a karaktersorozatok összehasonlítására használt `strcmp()` és `strncmp()` könyvtári függvények segítségével dolgozza fel:

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

void sorvegir( char *innev, char *outnev, char *s )
{
  int hossz = sorhossz( innev );
  char *sor = ( char * )malloc( ( hossz + 2 ) * sizeof( char ) );
  FILE *fin = fopen( innev, "r" ), *fout = fopen( outnev, "w" );
  while ( fgets( sor, hossz + 2, fin ) )
  {
    int sorh = strlen( sor ), szth = strlen( s ), diff = sorh - szth;
    if ( sor[ sorh-1 ] == '\n' && diff > 0 && !strncmp( s, sor + diff - 1, szth )
         ||
         sor[ sorh-1 ] != '\n' && diff >= 0 && !strcmp( s, sor + diff ) )
      fputs( sor, fout );
  }
  fclose( fin );
  fclose( fout );
  free( sor );
}
```

1.64. FELADAT. Írjon **programot**, amely egy **létező szöveges állomány** minden sorát **80** karakter hosszúságúra **egészíti ki szóközökkel**, ha rövidebbek a sorok 80 karakternél, és **csonkítja** a végén a sorokat, ha azok hosszabbak 80 karakternél! Az új sorokat egy **új** szöveges állományba kell írni.

A 80 karakternél rövidebb sorokat sok helyen ki lehet egészíteni szóközökkel, a leglátványosabb (és legkönnyebben ellenőrizhető) módon a sorok elején, ahogy azt a következő program is csinálja:

```
#include <stdio.h>

#define HATAR 80

int main()
{
  FILE *fin, *fout;
  char input[ 256 ], output[ 256 ], sor[ HATAR + 1 ];
  enum { BELUL, TUL, VEGE } allapot = BELUL;
  int darab = 0;

  printf( "Az input állomány: " );   scanf( "%s", input );
  printf( "Az output állomány: " );  scanf( "%s", output );
  fin = fopen( input, "r" );
  fout = fopen( output, "w" );

  while ( allapot != VEGE )
```

```
  {
    int ch = fgetc( fin );
    switch ( allapot )
    {
      case BELUL: if ( ch == EOF )
                  {
                    sor[ darab ] = '\0';
                    if ( darab != 0 )
                    {
                      int i;
                      for ( i = 0; i < HATAR - darab; ++i )
                        fputc( ' ', fout );
                      fprintf( fout, "%s", sor );
                    }
                    allapot = VEGE;
                  }
                  else if ( ch == '\n' )
                  {
                    int i;
                    sor[ darab ] = '\0';
                    for ( i = 0; i < HATAR - darab; ++i )
                      fputc( ' ', fout );
                    fprintf( fout, "%s\n", sor );
                    darab = 0;
                  }
                  else
                  {
                    sor[ darab++ ] = ch;
                    if ( darab == HATAR )
                    {
                      sor[ darab ] = '\0';
                      allapot = TUL;
                    }
                  }
                  break;
      case TUL:   if ( ch == EOF )
                  {
                    fprintf( fout, "%s", sor );
                    allapot = VEGE;
                  }
                  else if ( ch == '\n' )
                  {
                    fprintf( fout, "%s\n", sor );
                    allapot = BELUL;
                    darab = 0;
                  }
                  break;
    }
  }
  fclose( fout );
  fclose( fin );
}
```

1.65. FELADAT. Írjon **programot**, amely egy **létező** szöveges állomány sorainak mindegyikét **100 hosszúságúra** egészíti ki a sorok végén **szóközöket** szúrva be, ha rövidebb, illetve **elhagyva** a fölösleges karaktereket, ha hosszabb. Az új sorokat egy **most létrehozott** szöveges állományba kell elhelyezni.

A feladatot többféleképpen is meg lehet oldani.

(a) Első megoldásunkban állapotátmenet-gráfot használunk, hasonlóan a 1.64. feladatban szereplő

megoldáshoz.

```
#include <stdio.h>

#define HATAR 100

int main()
{
  FILE *fin, *fout;
  char input[ 256 ], output[ 256 ], sor[ HATAR + 1 ];
  enum { BELUL, TUL, VEGE } allapot = BELUL;
  int darab = 0;

  printf( "%s\n", __func__ );
  printf( "Az input állomány: " );   scanf( "%s", input );
  printf( "Az output állomány: " );  scanf( "%s", output );
  fin = fopen( input, "r" );
  fout = fopen( output, "w" );

  while ( allapot != VEGE )
  {
    int ch = fgetc( fin );
    switch ( allapot )
    {
      case BELUL: if ( ch == EOF )
                  {
                    sor[ darab ] = '\0';
                    if ( darab != 0 )
                    {
                      int i;
                      fprintf( fout, "%s", sor );
                      for ( i = 0; i < HATAR - darab; ++i )
                        fputc( ' ', fout );
                    }
                    allapot = VEGE;
                  }
                  else if ( ch == '\n' )
                  {
                    int i;
                    sor[ darab ] = '\0';
                    fprintf( fout, "%s", sor );
                    for ( i = 0; i < HATAR - darab; ++i )
                      fputc( ' ', fout );
                    fputc( '\n', fout );
                    darab = 0;
                  }
                  else
                  {
                    sor[ darab++ ] = ch;
                    if ( darab == HATAR )
                    {
                      sor[ darab ] = '\0';
                      allapot = TUL;
                    }
                  }
                  break;
      case TUL:   if ( ch == EOF )
                  {
```

```
                    fprintf( fout, "%s", sor );
                    allapot = VEGE;
                  }
                  else if ( ch == '\n' )
                  {
                    fprintf( fout, "%s\n", sor );
                    allapot = BELUL;
                    darab = 0;
                  }
                  break;
    }
  }
  fclose( fout );
  fclose( fin );
}
```

(b) Második megoldásunk – a 1.63. feladathoz hasonlóan – először meghatározza az input állomány leghosszabb sorának hosszát, majd ezt az adatot felhasználva soronként olvassa végig újra a feldolgozandó állományt.

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

#define max( a, b )   ( ( ( a ) > ( b ) ) ? ( a ) : ( b ) )

#define HATAR 100

int sorhossz( char *allomany )
{
  int hossz = 0, maxhossz = 0, ch;
  FILE *f = fopen( allomany, "r" );

  while ( ( ch = fgetc( f ) ) != EOF )
  {
    if ( ch == '\n' )
    {
      if ( hossz > maxhossz )
        maxhossz = hossz;
      hossz = 0;
    }
    else
      ++hossz;
  }
  if ( hossz > maxhossz )
    maxhossz = hossz;
  fclose( f );
  return maxhossz;
}

main()
{
  char input[ 256 ], output[ 256 ], *sor;
  int hossz;
  FILE *fin, *fout;
  printf( "Az input állomány: " );   scanf( "%s", input );
  printf( "Az output állomány: " );  scanf( "%s", output );
  hossz = sorhossz( input );
```

```
      sor = ( char * )malloc( max( hossz + 2, HATAR + 2 ) * sizeof( char ) );

      fin = fopen( input, "r" );
      fout = fopen( output, "w" );

      while ( fgets( sor, hossz + 2, fin ) )
      {
        int sorh = strlen( sor );
        int utolso = sor[ sorh - 1 ] != '\n';
        if ( !utolso )
          sor[ --sorh ] = '\0';
        if ( sorh < HATAR )
        {
          int i;
          for ( i = 0; i < HATAR - sorh; ++i )
            strcat( sor, " " );
        }
        else
          sor[ HATAR ] = '\0';
        if ( !utolso )
          strcat( sor, "\n" );
        fputs( sor, fout );
      }

      free( sor );

      fclose( fout );
      fclose( fin );
    }
```

1.66. FELADAT. Írjon **programot**, amely egy **magyar** szavakat tartalmazó **szöveges állomány** szavait ábécé sorrendben írja át egy **másik** állományba!

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

char *kodol( const char *s )
{
  char *kod = ( char * )malloc( ( strlen( s ) + 1 ) * sizeof( char ) ),
       *k = kod;
  while ( *s )
  {
    switch ( *s )
    {
      case 'a': case 'A':
        *k++ = 1;
        break;
      case 'á': case 'Á':
        *k++ = 2;
        break;
      case 'b': case 'B':
        *k++ = 3;
        break;
      case 'c': case 'C':
        if ( tolower( *( s + 1 ) ) == 's' )
        {
          *k++ = 5;
```

```
      ++s;
    }
    else if ( tolower( *( s + 1 ) ) == 'c' && tolower( *( s + 2 ) ) == 's' )
    {
      *k++ = 5;
      *k++ = 5;
      s += 2;
    }
    else
      *k++ = 4;
    break;
  case 'd': case 'D':
    if ( tolower( *( s + 1 ) ) == 'z' )
    {
      *k++ = 7;
      ++s;
    }
    else if ( tolower( *( s + 1 ) ) == 'd' && tolower( *( s + 2 ) ) == 'z' )
    {
      *k++ = 7;
      *k++ = 7;
      s += 2;
    }
    else if ( tolower( *( s + 1 ) ) == 'z' && tolower( *( s + 2 ) ) == 's' )
    {
      *k++ = 8;
      s += 2;
    }
    else if ( tolower( *( s + 1 ) ) == 'd' && tolower( *( s + 2 ) ) == 'z' &&
              tolower( *( s + 3 ) ) == 's' )
    {
      *k++ = 8;
      *k++ = 8;
      s += 3;
    }
    else
      *k++ = 6;
    break;
  case 'e': case 'E':
    *k++ = 9;
    break;
  case 'é': case 'É':
    *k++ = 10;
    break;
  case 'f': case 'F':
    *k++ = 11;
    break;
  case 'g': case 'G':
    if ( tolower( *( s + 1 ) ) == 'y' )
    {
      *k++ = 13;
      ++s;
    }
    else if ( tolower( *( s + 1 ) ) == 'g' && tolower( *( s + 2 ) ) == 'y' )
    {
      *k++ = 13;
      *k++ = 13;
      s += 2;
```

```
  }
  else
    *k++ = 12;
  break;
case 'h': case 'H':
  *k++ = 14;
  break;
case 'i': case 'I':
  *k++ = 15;
  break;
case 'í': case 'Í':
  *k++ = 16;
  break;
case 'j': case 'J':
  *k++ = 17;
  break;
case 'k': case 'K':
  *k++ = 18;
  break;
case 'l': case 'L':
  if ( tolower( *( s + 1 ) ) == 'y' )
  {
    *k++ = 20;
    ++s;
  }
  else if ( tolower( *( s + 1 ) ) == 'l' && tolower( *( s + 2 ) ) == 'y' )
  {
    *k++ = 20;
    *k++ = 20;
    s += 2;
  }
  else
    *k++ = 19;
  break;
case 'm': case 'M':
  *k++ = 21;
  break;
case 'n': case 'N':
  if ( tolower( *( s + 1 ) ) == 'y' )
  {
    *k++ = 23;
    ++s;
  }
  else if ( tolower( *( s + 1 ) ) == 'n' && tolower( *( s + 2 ) ) == 'y' )
  {
    *k++ = 23;
    *k++ = 23;
    s += 2;
  }
  else
    *k++ = 22;
  break;
case 'o': case 'O':
  *k++ = 24;
  break;
case 'ó': case 'Ó':
  *k++ = 25;
  break;
```

```
case 'ö': case 'Ö':
  *k++ = 26;
  break;
case 'ő': case 'Ő':
  *k++ = 27;
  break;
case 'p': case 'P':
  *k++ = 28;
  break;
case 'q': case 'Q':
  *k++ = 29;
  break;
case 'r': case 'R':
  *k++ = 30;
  break;
case 's': case 'S':
  if ( tolower( *( s + 1 ) ) == 'z' )
  {
    *k++ = 32;
    ++s;
  }
  else if ( tolower( *( s + 1 ) ) == 's' && tolower( *( s + 2 ) ) == 'z' )
  {
    *k++ = 32;
    *k++ = 32;
    s += 2;
  }
  else
    *k++ = 31;
  break;
case 't': case 'T':
  if ( tolower( *( s + 1 ) ) == 'y' )
  {
    *k++ = 34;
    ++s;
  }
  else if ( tolower( *( s + 1 ) ) == 't' && tolower( *( s + 2 ) ) == 'y' )
  {
    *k++ = 34;
    *k++ = 34;
    s += 2;
  }
  else
    *k++ = 33;
  break;
case 'u': case 'U':
  *k++ = 35;
  break;
case 'ú': case 'Ú':
  *k++ = 36;
  break;
case 'ü': case 'Ü':
  *k++ = 37;
  break;
case 'ű': case 'Ű':
  *k++ = 38;
  break;
case 'v': case 'V':
```

```
        *k++ = 39;
        break;
      case 'w': case 'W':
        *k++ = 40;
        break;
      case 'x': case 'X':
        *k++ = 41;
        break;
      case 'y': case 'Y':
        *k++ = 42;
        break;
      case 'z': case 'Z':
        if ( tolower( *( s + 1 ) ) == 's' )
        {
          *k++ = 44;
          ++s;
        }
        else if ( tolower( *( s + 1 ) ) == 'z' && tolower( *( s + 2 ) ) == 's' )
        {
          *k++ = 44;
          *k++ = 44;
          s += 2;
        }
        else
          *k++ = 43;
        break;
    }
    ++s;
  }
  *k = '\0';
  return kod;
}

void ekezettelenit( char *kod )
{
  while ( *kod )
  {
    switch ( *kod )
    {
      case 2:
        *kod = 1;
        break;
      case 10:
        *kod = 9;
        break;
      case 16:
        *kod = 15;
        break;
      case 25:
        *kod = 24;
        break;
      case 27:
        *kod = 26;
        break;
      case 36:
        *kod = 35;
        break;
      case 38:
```

```
        *kod = 37;
        break;
    }
    ++kod;
  }
}

int strcmphun( const char *s1, const char *s2 )
{
  char *kod1 = kodol( s1 ), *kod2 = kodol( s2 );
  int elteres = strcmp( kod1, kod2 ), ujelteres;
  if ( elteres == 0 )
  {
    free( kod1 );
    free( kod2 );
    return 0;
  }
  ekezettelenit( kod1 );
  ekezettelenit( kod2 );
  ujelteres = strcmp( kod1, kod2 );
  free( kod1 );
  free( kod2 );
  return ujelteres == 0 ? elteres : ujelteres;
}

main()
{
  FILE *in = fopen( "be.txt", "r" ), *out = fopen( "ki.txt", "w" );
  char szo[ 100 ], **tomb = NULL;
  int meret = 0, i, j;
  while ( fscanf( in, "%s", szo ) != EOF )
  {
    tomb = ( char ** )realloc( tomb, ++meret * sizeof( char * ) );
    tomb[ meret - 1 ] = ( char * )malloc( ( strlen( szo ) + 1 ) * sizeof( char ) );
    strcpy( tomb[ meret - 1 ], szo );
  }
  fclose( in );
  for ( i = meret - 2; i >= 0; --i )
    for ( j = 0; j <= i; ++j )
      if ( strcmphun( tomb[ j ], tomb[ j + 1 ] ) > 0 )
      {
        char *seged = tomb[ j ];
        tomb[ j ] = tomb[ j + 1 ];
        tomb[ j + 1 ] = seged;
      }
  for ( i = 0; i < meret; ++i )
    fprintf( out, "%s\n", tomb[ i ] );
  fclose( out );
}
```

1.67. FELADAT. Írjon **programot**, amely egy szöveges állományban elhelyezett, **szintaktikailag helyes** Pascal (C) forrásprogram szövegét úgy másolja át egy másik szöveges állományba, hogy közben kihagyja belőle a **megjegyzéseket**!

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

main()
```

```
{
  char input[ 256 ], output[ 256 ], *szoveg = NULL;
  FILE *fin, *fout;
  int allapot = 1;

  printf( "Az input állomány: " );   gets( input );
  printf( "Az output állomány: " );  gets( output );

  fin = fopen( input, "r" );
  fout = fopen( output, "w" );

  while ( allapot != 11 )
  {
    int ch = fgetc( fin ), hossz;
    switch ( allapot )
    {
      case 1:  if ( ch == EOF )
                 allapot = 11;
               else if ( ch == '\'' )
               {
                 fputc( ch, fout );
                 allapot = 2;
               }
               else if ( ch == '\"' )
               {
                 fputc( ch, fout );
                 allapot = 4;
               }
               else if ( ch == '/' )
               {
                 szoveg = malloc( 2 * sizeof( char ) );
                 strcpy( szoveg, "/" );
                 allapot = 6;
               }
               else
                 fputc( ch, fout );
               break;
      case 2:  fputc( ch, fout );
               if ( ch == '\'' )
                 allapot = 1;
               else if ( ch == '\\' )
                 allapot = 3;
               break;
      case 3:  fputc( ch, fout );
               allapot = 2;
               break;
      case 4:  fputc( ch, fout );
               if ( ch == '\"' )
                 allapot = 1;
               else if ( ch == '\\' )
                 allapot = 5;
               break;
      case 5:  fputc( ch, fout );
               allapot = 4;
               break;
      case 6:  if ( ch == '*' )
               {
                 free( szoveg );
```

```
                allapot = 7;
              }
              else if ( ch == '\\' )
              {
                hossz = strlen( szoveg );
                szoveg = realloc( szoveg, ( hossz + 2 ) * sizeof( char ) );
                strcat( szoveg, "\\" );
                allapot = 9;
              }
              else if ( ch == '/' )
                fputc( ch, fout );
              else
              {
                fprintf( fout, "%s%c", szoveg, ch );
                free( szoveg );
                szoveg = NULL;
                allapot = 1;
              }
              break;
      case 7:  if ( ch == '*' )
                allapot = 8;
              break;
      case 8:  if ( ch == '/' )
              {
                fputc( ' ', fout );
                allapot = 1;
              }
              else if ( ch == '\\' )
                allapot = 10;
              else
                allapot = 7;
              break;
      case 9:  hossz = strlen( szoveg );
              szoveg = realloc( szoveg, ( hossz + 2 ) * sizeof( char ) );
              szoveg[ hossz ] = ch;
              szoveg[ hossz + 1 ] = '\0';
              if ( ch != ' ' )
                allapot = 6;
              break;
      case 10: if ( ch != ' ' )
                allapot = 8;
              break;
    }
  }

  fclose( fin );
  fclose( fout );
}
```

1.68. FELADAT. Adva van egy **szöveges** állomány, amely egy **C főprogramot** tartalmaz. Írjon **programot**, amely az összes **azonosítót nagybetűsre** írja át, és az új főprogramot elhelyezi egy **másik szöveges** állományba!

Ha feltételezzük, hogy a feldolgozandó szöveges állomány nem tartalmaz az előfordítónak szóló (#-tel kezdődő) sorokat, akkor egy lehetséges megoldás a következő:

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
const char *kulcstomb[] = { "auto",     "double", "int",      "struct",
                            "break",    "else",   "long",     "switch",
                            "case",     "enum",   "register", "typedef",
                            "char",     "extern", "return",   "union",
                            "const",    "float",  "short",    "unsigned",
                            "continue", "for",    "signed",   "void",
                            "default",  "goto",   "sizeof",   "volatile",
                            "do",       "if",     "static",   "while" };
const int kulcsdarab = sizeof( kulcstomb ) / sizeof( char * );

int kulcsszo( char *s )
{
  int i;
  for ( i = 0; i < kulcsdarab; ++i )
    if ( !strcmp( kulcstomb[ i ], s ) )
      return 1;
  return 0;
}

char *pertelenito( char *s )
{
  char *uj = ( char * )malloc( ( strlen( s ) + 1 ) * sizeof( char ) );
  int i, j;
  for ( i = j = 0; i < strlen( s ); ++i )
    if ( isalnum( s[ i ] ) || s[ i ] == '_' )
      uj[ j++ ] = s[ i ];
  uj[ j ] = '\0';
  return uj;
}

int main()
{
  FILE *fin, *fout;
  char input[ 256 ], output[ 256 ], *szo;
  int allapot = 1;

  printf( "Az input állomány: " );   gets( input );
  printf( "Az output állomány: " );  gets( output );

  fin = fopen( input, "r" );
  fout = fopen( output, "w" );

  while ( allapot != 13 )
  {
    int ch = fgetc( fin ), hossz;
    switch( allapot )
    {
      case 1:  if ( ch == EOF )
                 allapot = 13;
               else if ( ch == '\"' )
               {
                 fputc( ch, fout );
                 allapot = 2;
               }
               else if ( ch == '\'' )
               {
                 fputc( ch, fout );
```

```
            allapot = 4;
          }
          else if ( ch == '/' )
          {
            fputc( ch, fout );
            allapot = 6;
          }
          else if ( isalpha( ch ) || ch == '_' )
          {
            szo = ( char * )calloc( 2, sizeof( char ) );
            szo[ 0 ] = ch;
            allapot = 11;
          }
          else
            fputc( toupper( ch ), fout );
          break;
case 2:   fputc( ch, fout );
          if ( ch == '\"' )
            allapot = 1;
          else if ( ch == '\\' )
            allapot = 3;
          break;
case 3:   fputc( ch, fout );
          allapot = 2;
          break;
case 4:   fputc( ch, fout );
          if ( ch == '\'' )
            allapot = 1;
          else if ( ch == '\\' )
            allapot = 5;
          break;
case 5:   fputc( ch, fout );
          allapot = 4;
          break;
case 6:   fputc( ch, fout );
          if ( ch == '\\' )
            allapot = 7;
          else if ( ch == '*' )
            allapot = 8;
          else if ( ch != '/' )
            allapot = 1;
          break;
case 7:   fputc( ch, fout );
          if ( ch != ' ' )
            allapot = 6;
          break;
case 8:   fputc( ch, fout );
          if ( ch == '*' )
            allapot = 9;
          break;
case 9:   fputc( ch, fout );
          if ( ch == '/' )
            allapot = 1;
          else if ( ch == '\\' )
            allapot = 10;
          else
            allapot = 8;
          break;
```

```
      case 10: fputc( ch, fout );
               if ( ch != ' ' )
                 allapot = 9;
               break;
      case 11: if ( ch == '\\' )
               {
                 hossz = strlen( szo );
                 szo = ( char * )realloc( szo, ( hossz + 2 ) * sizeof( char ) );
                 strcat( szo, "\\" );
                 allapot = 12;
               }
               else if ( isalnum( ch ) || ch == '_' )
               {
                 hossz = strlen( szo );
                 szo = ( char * )realloc( szo, ( hossz + 2 ) * sizeof( char ) );
                 szo[ hossz ] = ch;
                 szo[ hossz + 1 ] = '\0';
               }
               else
               {
                 char *uj = pertelenito( szo );
                 if ( kulcsszo( uj ) )
                   fprintf( fout, "%s", szo );
                 else
                 {
                   int i;
                   for ( i = 0; i < strlen( szo ); ++i )
                     fputc( toupper( szo[ i ] ), fout );
                 }
                 free( szo );
                 free( uj );
                 fputc( ch, fout );
                 allapot = 1;
               }
               break;
      case 12: hossz = strlen( szo );
               szo = ( char * )realloc( szo, ( hossz + 2 ) * sizeof( char ) );
               szo[ hossz ] = ch;
               szo[ hossz + 1 ] = '\0';
               if ( ch == '\n' )
                 allapot = 11;
               break;
    }
  }

  fclose( fin );
  fclose( fout );
}
```

1.69. FELADAT. Adva van egy szöveges állomány, amely egy szintaktikailag helyes, **egész** visszatérési értékkel rendelkező C **függvényt** tartalmaz. Írjon **programot**, amely meghatározza, hogy a függvény **rekurzív-e**! Az eredmény a **képernyőn** jelenjen meg!

A feladat a következő megfogalmazásokban is szerepelt a beugrókon:

Adva van egy **szöveges** állomány, amely egy szintaktikailag helyes, **valós** visszatérési értékkel rendelkező C (Pascal) **függvényt** tartalmaz. Írjon **programot**, amely eldönti, hogy a függvény **rekurzív-e**! Az eredmény jelenjen meg a **képernyőn**!

Adva van egy szöveges állomány, amely egy olyan szabályos **C függvényt** tartalmaz, amelynek **egy**

(1) formális paramétere van. Írjon egy **logikai függvényt**, amely akkor ad igaz értéket, ha a függvény **rekurzív**!

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define FALSE 0
#define TRUE  !FALSE

void betolt( FILE *f, char *puffer )
{
  int sztringben, karban, megjben, perperben, marad;
  char c, elozo = '\0', elozoelotti = '\0';
  sztringben = karban = megjben = perperben = marad = FALSE;
  for ( ; ; )
  {
    if ( ( c = fgetc( f ) ) == EOF )
      break;
    switch ( c )
    {
      case '*':
        if ( !sztringben && !megjben )
          if ( elozo == '/' )
            megjben = TRUE;
          else if ( elozo == '*' )
            *puffer++ = elozo;
        break;
      case '/':
        if ( elozo == '*' )
        {
          if ( megjben && !perperben )
            megjben = FALSE;
        }
        else if ( elozo == '/' )
          if ( !megjben && elozoelotti != '*' )
            megjben = perperben = TRUE;
        break;
      case '\n':
        if ( elozo == '*' && !sztringben && !megjben )
          *puffer++ = elozo;
        *puffer++ = c;
        if ( perperben )
          megjben = perperben = FALSE;
        break;
      case '"':
        if ( elozo == '*' && !sztringben && !megjben )
          *puffer++ = elozo;
        if ( elozo != '\\' && !karban && !megjben )
          sztringben = !sztringben;
        break;
      case '\'':
        if ( elozo == '*' && !sztringben && !karban && !megjben )
          *puffer++ = elozo;
        if ( elozo != '\\' && !sztringben && !megjben )
          karban = !karban;
        break;
      case '\\':
```

```
        if ( !sztringben && !karban && !megjben )
        {
          if ( elozo == '*' )
            *puffer++ = elozo;
          while ( ( c = fgetc( f ) ) == ' ' || c == '\t' )
            elozo = c;
          if ( c != '\n' )
          {
            ungetc( c, f );
            *puffer++ = c = elozo;
          }
          else
            marad = TRUE;
        }
        break;
      default:
        if ( !sztringben && !karban && !megjben )
        {
          if ( elozo == '/' && elozoelotti != '*' || elozo == '*' )
            *puffer++ = elozo;
          *puffer++ = c;
        }
    }
    if ( !marad )
    {
      elozoelotti = elozo;
      elozo = c;
    }
    else
      marad = FALSE;
  }
  *puffer = '\0';
}

char *fvnev_keres( char *puffer )
{
  char *fvnev = ( char * )malloc( 256 * sizeof( char ) ), *pufmut, *nevmut;
  for ( pufmut = puffer; *pufmut != '('; ++pufmut )
    ;
  while ( strchr( " \t\n", *--pufmut ) )
    ;
  while ( --pufmut >= puffer && ( isalnum( *pufmut ) || *pufmut == '_' ) )
    ;
  for ( nevmut = fvnev; !strchr( " \t\n(", *nevmut = *++pufmut ); ++nevmut )
    ;
  *nevmut = '\0';
  return fvnev;
}

int rekurziv( char *puffer, char *fvnev )
{
  char *pufmut = puffer, *nevmut;
  long int pufhossz = strlen( puffer ), nevhossz = strlen( fvnev );
  while ( *++pufmut != '{' )
    ;
  while ( pufmut <= puffer + pufhossz - nevhossz )
  {
    for ( nevmut = fvnev; *nevmut && *pufmut == *nevmut; ++pufmut, ++nevmut )
```

```
      ;
    if ( !*nevmut )
    {
      char *seged = pufmut;
      while ( strchr( " \t\n", *seged++ ) )
        ;
      if ( *--seged == '(' )
      {
        seged = pufmut - nevhossz;
        if ( !isalnum( *--seged ) && *seged != '_' )
          return TRUE;
      }
    }
    pufmut -= nevmut - fvnev - 1;
  }
  return FALSE;
}

main()
{
  FILE *fajl;
  long int meret;
  char allnev[ 256 ], *puffer, *fvnev;
  printf( "Kérem az állomány nevét: " );  scanf( "%s", allnev );
  fajl = fopen( allnev, "r" );
  if ( !fajl )
  {
    fprintf( stderr, "Hiba az állomány megnyitásánál!\n" );
    exit( 1 );
  }
  fseek( fajl, 0, SEEK_END );
  meret = ftell( fajl );
  fseek( fajl, 0, SEEK_SET );
  puffer = ( char * )malloc( meret + 1 );
  if ( !puffer )
  {
    fprintf( stderr, "Hiba a memóriafoglalásnál!\n" );
    fclose( fajl );
    exit( 2 );
  }
  betolt( fajl, puffer );
  fclose( fajl );
  puts( puffer );
  fvnev = fvnev_keres( puffer );
  printf( "Függvénynév: *%s*\n", fvnev );
  if ( rekurziv( puffer, fvnev ) )
    puts( "Rekurzív." );
  else
    puts( "Nem rekurzív." );
  free( fvnev );
  free( puffer );
}
```

2. fejezet

Feladatsorok

2.1. 1994. május 16., nappali tagozat

1. Készítsen **eljárást**, amely a standard bemenetről beolvas egy legfeljebb 10×10-es méretű, egész elemekből álló négyzetes mátrixot, és eldönti, hogy az háromszög mátrix-e! Az eljárásnak van egy input paramétere, amely a mátrix aktuális méretét adja meg, és egy logikai típusú output paramétere, amely akkor igaz, ha a mátrix háromszögmátrix.

2. Adva van egy `szoveg.txt` nevű szöveges állomány, amely soraiban magyar szavak állnak egymástól egy szóközzel elválasztva. Az utolsó szó után nem áll szóköz, hanem közvetlenül a sorvége jel következik. Írjon **programot**, amely végigolvassa az állományt, és

   (a) megszámolja az állományban lévő szavakat;

   (b) létrehoz egy másik szöveges állományt, amelybe a **magas**, **mély**, **vegyes** szavak valamelyikét helyezi el annak megfelelően, hogy az adott szó milyen hangrendű. Az új állományban annyi sor és minden sorban annyi szó legyen, mit az eredetiben volt.

3. Adva van egy egészeket tartalmazó típusos állomány. A mérete akkora, hogy nem fér be a tárba. Rendezze az állományt a saját helyén. Segédállományt nem használhat, segédváltozókat igen.

4. Írjon egy olyan logikai **függvényt**, amely programozási nyelvek (Pascal, C, Ada, LISP, COBOL, ALGOL, ...) egy halmazát definiálja, majd eldönti, hogy a paramétereként megadott programozási nyelv benne van-e a halmazban. Használjon felsorolásos típust!

5. Adott egy állomány, amelyben az egyes rekordok különböző típusú adatokat tartalmaznak. A rekordok szerkezete a következő:

   1. bájt: a tartalmazott adatok típusa
      - 0: integer
      - 1: real
      - 2: boolean
      - 3: char

   2. bájt: az adott típusú adatok száma (max. 255),
   3. és további bájtok: az adatok.

   Válogassuk le az állományból a karaktereket tartalmazó rekordokat és hozzunk létre belőlük egy szöveges állományt `szo.txt` néven. Ha már létezik ilyen nevű állomány, akkor bővítsük azt.

2.2. 1995. május 15., nappali tagozat

1. Tekintsük az alábbi láncolt listát, amely `node` típusú elemekből áll:

```
type
  node_m = ^node;
  node   = record
             value: real;
             next:  node_m;
           end;
```

Írjon olyan Turbo Pascal függvényt, amely két ilyen listán kompozíciós szorzást végez, azaz az eredménye egy olyan lista, amelyben az első elem a két lista első elemének szorzata, a második elem a két második elem szorzata stb. Ha az egyik lista rövidebb, mint a másik, akkor feltételezzük, hogy a hátralévő elemei nullák. A függvény egy mutatót adjon vissza a szorzatlista első elemére (fej). A függvény két paramétere mutató az első, illetve a második lista első elemére (fej):

```
function listmult( list1, list2: node_m ): node_m;
```

2. Írjon olyan hash függvényt, amely egy paraméterként átvett sztringre előállít egy számot (címet) 1 és N között. A függvény vegye figyelembe a sztringben előforduló konkrét karaktereket és a sztring hosszát. A függvény első paramétere a sztring, a második az N szám legyen, a visszaadott érték pedig a cím:

```
function hash( key: string; N: longint ): longint;
```

3. Tekintsünk egy keresőfát az alábbi csomópont-szerkezettel:

```
type
  node_m = ^node;
  node   = record
             value: integer;
             left:  node_m;
             right: node_m;
           end;
```

A `left` a bal oldali, a `right` a jobb oldali részfára mutat.

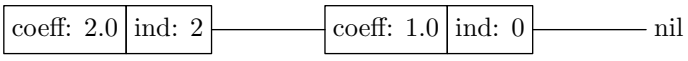
A bal oldali részfában lévő elemek kisebbek a gyökérben lévő elemnél (`value`), a jobb oldali részfában lévő elemek nagyobbak. Írjon olyan függvényt, amely meghatározza, hogy a paraméterként kapott egész hányadik szinten van a fában. A gyökér szintje legyen 0. A visszaadott érték a szintszám. Ha a keresett érték nem fordul elő a fában, a visszaadott érték legyen -1.

```
function depth( x: integer ): integer;
```

4. Egy valós polinomot a következőképpen ábrázolunk láncolt listában: a lista elemei a következő típusúak:

```
type
  element_m = ^element;
  element   = record
                coeff: real;
                ind:   integer;
                next:  element_m;
              end;
```

Az `ind` a kitevőt, a `coeff` az adott taghoz tartozó együtthatót jelenti. Például a $2x^2+1$ polinomot a következőképpen ábrázoljuk:



Írjon olyan függvényt, amelyik a polinom helyettesítési értékét kiszámolja! A függvény első paramétere mutató az első elemre (fej), a második pedig pedig az a hely, ahol a helyettesítési értéket venni kell.

```
function value( head: element_m; x: real ): real;
```

5. Készítsen egy függvényt, amely kiértékel egy kifejezésfát! A kifejezésfa levélelemei tartalmazzák a numerikus konstansokat, míg a további csomópontok a műveleti jeleket (négy alapművelet: `'+'`, `'-'`, `'*'`, `'/'`). A függvény paraméterként egy mutatót kap a fa gyökérelemére, függvényértékként pedig a kifejezés értékét szolgáltatja. Feltesszük, hogy a megadott kifejezés helyes.

```
type
  pnode = ^node;
  node  = record
            left:  pnode;
            right: pnode;
            case boolean of
              true:  ( value:  real; );
              false: ( opcode: char; );
          end;

function eval( root: pnode ): real;
```

6. Készítsen egy függvényt, amely eldönti egy négyzetes mátrixról, hogy háromszögmátrix-e, illetve hogy diagonális mátrix-e! A függvény paraméterként egy mutatót és egy méretet kap. A méret paraméter a mátrix sorainak (és oszlopainak) a számát tartalmazza, a mutató pedig egy egydimenziós egész elemű tömbre mutat. A mátrix elemei ezen tömbben helyezkednek el sorfolytonosan.

```
const
  maxmatrix = 100;
type
  mtipus  = ( also_haromszog, felso_haromszog, diagonalis, egyeb );
  pmatrix = ^array[ 1..maxmatrix*maxmatrix ] of integer;

function matrix_teszt( matrix: pmatrix; meret: integer ): mtipus;
```

7. Készítsen egy függvényt, amely egy egész számot átalakít egy adott számrendszerbeli sztringgé! A függvény első paramétere az átalakítandó szám, második paramétere a számrendszer alapja, a visszatérési érték pedig az eredmény sztring. A számrendszer alapszáma a $2, \ldots, 35$ intervallumba esik.

```
function konvertal( mit: longint; alap: integer ): string;
```

2.3. Időpontja ismeretlen, nappali tagozat

1. Tekintsük az alábbi szerkezettel adott rendezett bináris fát!

```
type
  str       = string[ 20 ];
  node_m    = ^node;
  node      = record
                job_nev: str;
                memory:  longint;
                next:    node_m;
              end;
  joblist_m = ^joblist;
  joblist   = record
                job_nev: str;
                next: joblist_m;
              end;
```

A fa a `job_nev` mező alapján van rendezve növekvő sorrendben.

Írjon olyan Turbo Pascal függvényt, amely egy láncolt listában növekvő sorrendben visszaadja azokat a `job_nev` értékeket, amelyekre a `memory` értéke egy adott `x` értéknél nagyobb vagy egyenlő. A függvény első paramétere a fa gyökere, a második paramétere pedig az `x`. A visszaadott lista szerkezete: `joblist`.

```
function usage( root: node_m; x: longint): joblist_m;
```

2. Írjon olyan eljárást, amely az alábbi szerkezetű listával megvalósított veremben csak a benne előforduló legnagyobb számot hagyja meg. A veremben különböző számok vannak.

```
type
  stack_m = ^stack;
  stack   = record
              x:    longint;
              next: stack_m;
            end;
```

Az eljárás paramétere: mutató a verem tetejére.

```
procedure max( top: stack_m );
```

3. Mutasson a **ptr** nevű mutató egy 52 elemű `integer` típusú elemekből álló memóriaterületre. Tekintsük az alábbi szerkezetű láncolt listát!

```
type
  lapt   = 1..52;
  node_m = ^node;
  node   = record
             mit  : lapt;
             mivel: lapt;
             next : node_m;
           end;
```

E lista elemei egy cserét határoznak meg a fenti (`ptr`) tömbben. A `mit` indexű elemet fel kell cserélni a `mivel` indexűvel. Írjon egy olyan eljárást, amely az összes, a listában felsorolt cserét elvégzi a megadott tömbben. Az eljárás első paramétere mutató a tömb (`ptr`) első elemére, a második paraméter pedig mutató a lista első elemére.

```
procedure csere( ptr: ^integer; list: node_m );
```

4. Adva van egy szövegfájlban egy valósakból álló zárt intervallumsorozat. Minden intervallum új sorban van, a két végpont pedig vesszővel van elválasztva. Például:

```
12.45, 12.55
1.0, 2.5
```

Írjon olyan Turbo Pascal eljárást, amely a képernyőre kiírja azt a legbővebb intervallumot, amely egy adott pontot tartalmaz, de nincs a felsorolt intervallumokkal egyetlen közös pontja sem. Az eljárás első paramétere a szövegfájl neve, a második paramétere pedig a kérdéses pont.

```
procedure maxi( fnev: string; point: real );
```

5. Készítsen egy olyan függvényt, amely egy tárban ábrázolt, `integer` típusú elemekből álló mátrixról eldönti, hogy szimmetrikus-e! Ha szimmetrikus, a visszaadott érték `true`, ellenkező esetben `false`.

A mátrix egy összefüggő memóriaterületen van sorfolytonosan tárolva. A függvény paramétere a mátrix első sorában lévő első elemnek a címe.

```
function szimmetrikus( mx: ^integer ): boolean;
```

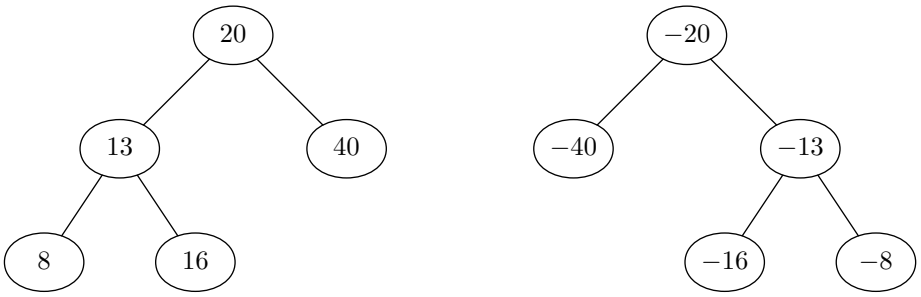
6. Adva van a memóriában egy bináris keresési fa az alábbi csomópont-szerkezettel:

```
type
  key_t  = integer;
  node_m = ^node;
  node   = record
             key:  key_t;
             next: node_m;
           end;
```

Írjon olyan Turbo Pascal eljárást, amely minden csomópont bal és jobb oldali gyermekét felcseréli és ugyanakkor a `key` mezőben található egészeket kicseréli a -1-szeresükre (`-key`). (Az eredmény szintén egy bináris fa.) Az eljárás paramétere a fa gyökere.

```
procedure swaptree( root: node_m );
```

Példa:



2.4. Időpontja ismeretlen, nappali tagozat

1. Írjon az alábbi prototípussal megegyező függvényt:

   ```
   char *concat( int db, ... );
   ```

   A függvény működése: az első változóban megadott darabszámú `char *` típusként átvett változók által meghatározott karakterláncokat fűzze össze, és az új karakterlánc kezdőcímét adja vissza.

2. Írjon az alábbi prototípussal megegyező függvényt:

   ```
   int include( FILE *inp, FILE *out );
   ```

   A függvény működése: a preprocesszor hasonló funkcióját látja el. Az első paraméterben megadott állományból a `#include <állománynév>` alakú sorokat helyettesíti a megadott állomány tartalmával. Az új állomány is tartalmazhat hivatkozásokat. Az így kifejtett szöveg a második paraméterben megadott állományba kerüljön. A függvény visszatérési értéke 0, ha sikeres volt a végrehajtás; 1, ha megnyitási hiba történt; 2, ha újból meg akarunk nyitni egy már használatban lévő állományt (végtelen ciklus).

3. Írjon olyan függvényt, amely egy paraméterben megkapott fájlról eldönti, hogy 8 bites-e, azaz a bájtjainak legmagasabb helyiértékű bitje magas-e vagy nem. (Ha egy bájt 8 bites, akkor az egész fájl 8 bites.) Ha a fájl 8 bites, akkor 1-et, különben 0-t ad vissza. Prototípus:

   ```
   int eight( FILE *fp );
   ```

4. Készítsen függvényt, amely beolvas egy X11 bitmap fájlt a memóriába. A bitmap számára a függvény foglaljon helyet. A függvény prototípusa:

   ```
   void *readxbm( FILE * );
   ```

   Az X11 bitmap formátum: a fájl tartalmaz két `#define` sort, amelyek a bitmap szélességét és magasságát adják meg. Ezt követi egy C szintaktikájú kezdőértékkel ellátott tömb deklaráció. A bitmap minden pontot egy biten ábrázol, soronként bájthatárra igazítva. Elnevezések:

   ```
   #define filename_width  100
   #define filename_height 150
   static char filename_bits[] = { 0x12, 0x34, ... };
   ```

5. Készítsen függvényt, amely veremkalkulátort valósít meg! A függvény bemenő paramétere az input állomány, visszatérési értéke egy valós szám, a műveletek eredménye. Amennyiben a fájlból számot olvas a függvény, az bekerül egy verembe, amennyiben műveleti jelet, úgy azt végre kell hajtani a verem tetején található két értékkel, és az eredmény visszakerül a verembe. A függvény visszatérési értéke az a szám, amely az input fájl végének elérésekor a verem tetején található. Minimálisan a négy alapműveletet kell implementálni, s tekintettel kell lenni a negatív számokra.

```
float scale( FILE * );
```

6. Készítsen függvényt, amely paraméterben megkapott bitképbe téglalapot rajzol. A képben egy képpont 1 bites, 0 vagy 1 értékű lehet. A téglalap rajzolása során a téglalap oldalaihoz tartozó pontokat 1 értékűre kell állítani. A függvény első paramétere a kép címe a memóriában, a második a kép mérete képpontokban x irányban, a harmadik az y irányú méret, a negyedik és ötödik a bal felső sarok, a hatodik és hetedik a téglalap jobb alsó sarkának koordinátái.

```
Rec( char *kep, int xw, int yw, int x1, int x2, int y1, int y2 );
```

2.5. 1996. május 18., nappali tagozat

1. Egy **bináris fa tükrözésének** nevezzük a következőt: a fa gyökerétől indulva, szintenként megcseréljük minden elem bal és jobb oldali részfáját.

   Írjon egy **logikai függvényt**, amely paraméterként megkap egy **egészeket** tartalmazó bináris fa gyökerét címző **mutatót**, **tükrözi** a fát, és **eldönti**, hogy a tükrözés után kapott fa **megegyezik-e** az induló fával!

2. Egy szöveges állományban a sorokban **postfix** alakú kifejezések vannak adva. A kifejezésekben a $+$, $-$, $*$, / műveleti jelek és előjel nélküli **egész** számok mint operandusok szerepelnek. A műveleti jeleket és az operandusokat egy-egy szóköz választja el egymástól.

   Írjon egy **eljárást**, amely paraméterként megkapja az **állomány nevét**, és **kiértékeli** a kifejezéseket, ezek értékét pedig elhelyezi egy **típusos** állományban!

3. Adva van egy **típusos** állomány a következő **rekordszerkezettel**:

kulcs	–	egész
név	–	szöveges
lakcím	–	szöveges

   Írjon egy **eljárást**, amely bemenő paraméterként megkapja az **állomány nevét**! Az állomány rekordjaiból hozzon létre egy **keresőfát**! Kimenő paraméter legyen a **fa gyökerét címző mutató**!

4. Adva van egy **szöveges állomány**, amely **maximum 10** hosszúságú, az **angol** ábécé betűiből álló, egymástól **1 szóközzel** elválasztott szavakat tartalmaz.

   Írjon **programot**, amely a szavakat egy **egyirányban láncolt rendezett listában** helyezi el! Írassa ki **képernyőre** azt a **szót** és az **előfordulás számát**, amely a leggyakrabban fordul elő az állományban (vigyázat, több ilyen is lehet)!

5. Írjon egy **eljárást**, amely bemenő paraméterként kap egy **valós** elemekből felépített **bináris fa** gyökerét címző **mutatót**! A bináris fához készítse el a **hierarchikus listánál** megismert **zárójeles reprezentációt**, és helyezze el azt egy **szöveges** állományban!

6. Írjon egy **eljárást**, amely egy **valósakat** tartalmazó **tetszőleges kétdimenziós** tömb sorait úgy rendezi át, hogy azok a sorokban elhelyezett elemek **átlaga** szerint **növekvően** helyezkedjenek el!

7. Írjon egy **hash függvényt**, amelynek bemenő paraméterei egy **sztring** és egy **egész** szám. A függvény határozzon meg **egy számot** 1 és az egész szám között, a meghatározásnál vegye figyelembe a sztring **hosszát** és a sztringben előforduló **konkrét karaktereket**!

2.6. 1996. június 13., nappali tagozat

1. Egy **bináris fa tükrözésének** nevezzük a következőt: a fa gyökerétől indulva, szintenként megcseréljük minden elem bal és jobb oldali részfáját.

   Írjon egy **logikai függvényt**, amely paraméterként megkap egy **egészeket** tartalmazó bináris fa gyökerét címző **mutatót**, **tükrözi** a fát, és **eldönti**, hogy a tükrözés után kapott fa **megegyezik-e** az induló fával!

2. Egy szöveges állományban a sorokban **postfix** alakú kifejezések vannak adva. A kifejezésekben a $+$, $-$, $*$, / műveleti jelek és előjel nélküli **egész** számok mint operandusok szerepelnek. A műveleti jeleket és az operandusokat egy-egy szóköz választja el egymástól.

   Írjon egy **eljárást**, amely paraméterként megkapja az **állomány nevét**, és minden kifejezéshez **felépíti** a megfelelő (szétszórt módon ábrázolt) bináris fát, továbbá a fák gyökereinek címét **elhelyezi** egy **egyirányban láncolt listában**. Az eljárás kimenő paramétere a **lista első elemét címző mutató** legyen.

3. Adva van egy **típusos** állomány a következő **rekordszerkezettel**:

kulcs	–	egész
név	–	szöveges
lakcím	–	szöveges

   Írjon egy **eljárást**, amely bemenő paraméterként megkapja az **állomány nevét**! Az állomány rekordjaiból hozzon létre egy **rendezett táblázatot**! Kimenő paraméter legyen a táblázat elemeinek **száma** és (reprezentációtól függően) a **táblázat** vagy a **táblázat első elemét címző mutató**!

4. Adva van egy **szöveges állomány**, amely **maximum 10** hosszúságú, az **angol** ábécé betűiből álló, egymástól **1 szóközzel** elválasztott szavakat tartalmaz.

   Írjon **programot**, amely meghatározza, hogy az állományban **hány különböző szó** van, és ezekhez megállapítja az előfordulásuk **gyakoriságát** is! A különböző szavakat a gyakoriságokkal együtt egy **típusos állományban** kell elhelyezni!

5. Írjon egy **függvényt**, amely bemenő paraméterként kap egy **valós** elemekből felépített **keresőfa gyökerét címző mutatót** és egy **valós** értéket! A függvény határozza meg, hogy az adott érték a fa **hányadik szintjén** fordul elő! A gyökér szint a 0. szint. Ha az érték nincs a fában, a visszaadott érték: -1.

6. Írjon egy **eljárást**, amely egy **valósakat** tartalmazó **kvadratikus mátrix** főátló alatti, főátló feletti és főátlóban álló elemeinek az **összegét** határozza meg!

7. Adva van egy **bináris fa folytonos reprezentációja** három egydimenziós tömbben (`adat`, `bal`, `jobb`). Az első elem a gyökérelem.

   Írjon egy **eljárást**, amely bemenő paraméterként megkapja a **fa elemeinek számát** és a **három tömböt**. Az eljárás **állítsa elő** a fa **szétszórt reprezentációját**, és kimenő paraméterként adja meg a **gyökér címét**!

2.7. 1996. december 14., nappali tagozat

1. Írjon olyan programot, amely két állomány tartalmát hasonlítja össze, és az első különbség pozícióját a standard hibakimenetre írja! Amennyiben a két állomány tartalma azonos, a program nem ír ki semmit. A program egy vagy két állománynevet kaphat futásidejű paraméterként (argumentumként). Ha egy paramétert kapott, akkor a standard bemenetet kell tekinteni a másik hasonlítandó állományként.

2. Írjon olyan programot, amely egy állománynevet kap futásidejű paraméterként (argumentumként). Ezen állomány tartalmát olvassa be egy kétirányban láncolt listába. Ezután jelenítse meg a listába felolvasott állomány 24 sorát a képernyőn úgy, hogy a 'B' billentyű lenyomására egy sorral visszafelé, az 'N' hatására egy sorral előrefelé lehessen „lapozni". Kilépés a 'Q' billentyűvel.

   Megjegyzés: az állomány sorai maximum 80 karakter hosszúak.

3. Írjon olyan függvényt, amely két állománynevet kap paraméterként. Az egyik állomány (alapállomány) alkalmazottak adatait tárolja rendezetlenül. A másik állomány (indexállomány) egy-egy rekordja az alapállomány rekordjainak elsődleges azonosítóját és az adott alaprekord fizikai pozícióját tartalmazza. A függvény feladata – az indexállomány felhasználásával – az alapállomány rekordjainak rendezett kiíratása a képernyőre.

   Az alapállomány rekordszerkezete:

```
struct employee
{
  char name[ 50 ];
  char pid[ 12 ];   /* személyi azonosító, elsődleges kulcs */
  long salary;
};
```

   Az indexállomány rekordszerkezete:

```
struct employee
{
  char pid[ 12 ];
  long pos;
};
```

4. Írjon olyan függvényt, amelynek prototípusa a következő:

```
char *char_dup( char *str, char c );
```

   A függvény feladata a kapott karakterláncban a második paraméterként megadott karakter összes előfordulásának megduplázása, s az eredmény elhelyezése egy megfelelő méretű, a függvény által allokált karakterláncban. A paraméterként kapott karakterlánc tartalma maradjon változatlan.

   A függvény visszatérési értéke a létrehozott karakterlánc címe legyen, illetve `NULL`, ha nem sikerült a memóriafoglalás!

5. Írjon olyan függvényt, amely egy bináris fában megadott kifejezést értékel ki! A kifejezésekben csak a négy alapműveletre ($+$, $-$, $*$, $/$) kell számítani. A kifejezésekben szereplő numerikus értékek előjeles egész számok. A kifejezés megadására szolgáló fa egy csomópontjának típusa:

```
struct node
{
  char operator;
  int value;
  struct node *right;
  struct node *left;
};
```

   A struktúra `operator` nevű mezőjében vagy a műveleti jel (`'+'`, `'-'`, `'*'`, `'/'`), vagy egy szóköz szerepel, ez utóbbi esetben a struktúra `value` mezőjének tartalma egy egész szám (ekkor a csomópont levélelem). A függvény visszatérési értéke a fában megadott kifejezés értéke legyen!

   A függvény prototípusa:

```
float evaluate( struct node *expression );
```

2.8. 1997. május 26., nappali tagozat

1. Adva van egy `feladat1.txt` nevű **szöveges** állomány, amelynek soraiban **magyar** szavak helyezkednek el úgy, hogy minden szó után áll egy **szóköz**. Az állomány sorainak száma tetszőleges, egy sorban akárhány szó lehet, a szavak maximális hossza **15**.

   Írjon **programot**, amely meghatározza az állományban előforduló **különböző** szavak **előfordulási gyakoriságát**! Feltesszük, hogy **legfeljebb 500** különböző szó lehet. Az eredményt `szo`, `gyak` felépítésű rekordokban helyezze el a `feladat2.dat` nevű **típusos** állományban!

2. Az előző feladat eredmény állományát felhasználva írjon olyan **függvényt**, amely megadja a **leggyakoribb** szavakat!

3. Az első feladat eredmény állományát felhasználva írjon olyan **függvényt**, amely megadja az ábécé sorrendben **legnagyobb** szót!

4. Adva van a `feladat3.txt` szöveges állomány, amely egy **Pascal programot** tartalmaz. Írjon **eljárást**, amely a program szövegéből eltávolítja a **megjegyzéseket**, és az eredményt helyezze a `feladat4.txt` szöveges állományba!

5. Billentyűzetről megadunk egy **teljesen zárójelezett kifejezést**, amely csak egyetlen betűből álló azonosítójú változókat és a $+$, $-$, $*$, / kétoperandusú operátorokat tartalmazhatja a zárójeleken kívül. Írjon **programot**, amely beolvassa a kifejezést, és ha az szintaktikusan helyes, akkor felépíti a hozzátartozó **bináris fát**, ha pedig hibás, akkor képernyőre írja azt a részkifejezést, amely még hibátlan volt, és ezután megjelöli a hiba okát! Feltesszük, hogy redundáns zárójelek nincsenek.

6. Írjon egy **eljárást**, amely az előző feladat szintaktikusan helyes kifejezését képernyőre írja és megjelöli azt az operátort, amely **utoljára** lesz végrehajtva a kifejezés kiértékelésénél!

2.9. 1997. június 10., nappali tagozat

1. Írjon egy **eljárást**, amely bemenő paraméterként megkap egy **létező szöveges állomány** nevet és egy **sztringet**. Az állomány sorainak száma tetszőleges, egy sor **maximálisan 999** karakterből állhat. Az eljárás határozza meg a **legrövidebb** és **leghosszabb** sorokat, és ezeket írja egy szövegállományba, továbbá döntse el, hogy **van-e** az állományban olyan sor, amely **megegyezik** a megadott sztringgel!

2. Adva van egy egészeket tartalmazó **egyirányban láncolt lista**. Írjon **eljárást**, amely megkapja bemenő **paraméterként** a **lista fejét** és egy **verembe** helyezi (a listabeli előfordulás sorrendjében) azokat at elemeket, melyeknek a listaelemek átlagától való eltérése kisebb mint 2!

3. Adva van egy **Pascal azonosítókat** tartalmazó szöveges állomány. Az azonosítókat **vessző** választja el. Írjon **programot**, amely az azonosítókból létrehoz egy olyan **keresőfát**, ahol a csomópontok szerkezete: **azonosító**, **gyakoriság**. Adja meg a **leggyakrabban** előforduló azonosítót!

4. Írjon egy **logikai függvényt**, amely egy billentyűzetről megadott karaktersorozatról eldönti, hogy az **szabályos Pascal valós konstans-e**!

5. Egy **szöveges állományban** az egyes sorokban **szabályos prefix** alakú kifejezések vannak, amelyekben a $+$, $-$, $*$, / operátorok és egész számok mint operandusok szerepelnek. Az operátorokat és operandusokat egymástól egy **szóköz** választja el. Írjon **programot**, amely kiértékeli a kifejezéseket, és ezek eredményét elhelyezi egy **típusos** állományban!

2.10. 1997. június 16., nappali tagozat

1. Írjon egy **sztring** típusú **függvényt**, amely bemenő paraméterként megkapja egy kifejezés fájának gyökerét címző mutatót, és előállítja a kifejezés **teljesen zárójelezett** alakját!

2. Írjon egy **eljárást**, amely bemenő paraméterként megkap egy **létező szöveges állomány** nevet és egy **sztringet**! Az állomány sorainak száma tetszőleges, egy sor **maximálisan** 512 karakterből állhat. Az eljárás hozzon létre egy egyirányban láncolt listát azon sorokból, amelyeknek a **vége megegyezik** a megadott sztringgel! A további sorokat írja egy másik szöveges állományba!

3. Adva van egy **típusos állomány a következő** rekordszerkezettel:

SOR	–	byte
OSZLOP	–	byte
ERTEK	–	real

Az állomány egy ritka mátrix **háromsoros oszlopfolytonos** reprezentációját tartalmazza. Az **első rekord** SOR és OSZLOP értéke a mátrix **méretét** adja meg. Írjon **eljárást**, amely bemenő paraméterként megkapja az állomány nevét, és képernyőre írja a mátrix azon **sorát**, amely a legtöbb nem-nulla elemet tartalmazza (feltesszük, hogy csak egy ilyen van)!

4. Írjon **függvényt**, amely egy `string[12]` típusú elemeket tartalmazó **veremből** eltávolítja azokat az elemeket, amelyek **nem Pascal azonosítók**!

5. Egy **nem bináris** fa elemeit reprezentáljuk egy **sztring** típusú adatelemmel és egy **ötelemű** mutatóvektorral. Írjon **mutató** típusú **függvényt**, amely paraméterként megkapja a fa gyökérmutatóját, és előállítja annak **bináris** reprezentációját! A visszaadott érték a bináris fa gyökerét címezi.

2.11. 1998. május 18., nappali tagozat

1. Adva van egy szövegállomány, amely egy szintaktikusan helyes Pascal programot tartalmaz. A programban alprogramok nincsenek, és egyetlen `var` utasítás szerepel benne. Írjon **programot**, amely megadja, hogy a deklarált **változókra** mely **sorokban** történik hivatkozás a szövegben. Az output a **képernyőn** keletkezzen! A sorokat 1-től kezdődően sorszámozza!

2. Írjon egy **függvényt**, amely paraméterként egy **nemüres bináris fa** gyökérmutatóját kapja meg, és visszaadja, hogy melyik a **legalacsonyabb szintszámú** olyan elem, amelynek nincs baloldali rákövetkezője! A gyökér szintszáma 0.

3. Írjon **programot**, amely **billentyűzetről** megadott **kifejezésekről** eldönti, hogy **szabályos Pascal** kifejezések-e! Minden kifejezést egy **Enter** zár le, benne szóköz nem lehet. A kifejezés sorainak végét egy **szóköz Enter** sor begépelése jelzi. A kifejezés operandusai **egybetűs azonosítók** és előjel nélküli **egész számok** lehetnek. Operátorok: $+$, $-$, $*$, $/$. **Zárójelek** tetszőlegesen alkalmazhatók. A program írja a képernyőre a kifejezést, ha az szabályos, akkor adja ezt az üzenetet, egyébként jelölje meg azt az elemet, ami a hibát okozza és írjon ki hibaüzenetet!

4. Adva van egy szövegállomány, benne **egy szóközzel elválasztott** (nem feltétlenül különböző) **szavak**. Két szó „hangzási távolságán” értjük a következőt: a szavakat az elejükön kezdve összehasonlítjuk karakterenként. Eltérő karakterek esetén a résztávolság értéke 1, ha azonos pozíción azonos mássalhangzó van, akkor 0.5, azonos magánhangzó esetén 0.1. A távolságot a résztávolságok összege adja.

   Írjon **eljárást**, amely paraméterként megkapja az állomány **nevét**, és egy **szót**, majd egy output állományban elhelyezi azt a **tíz** szót, amely **legközelebb** van a megadott szóhoz! A szavak mellett szerepeljen a **távolság** is.

5. Adva van egy szöveges állomány, amely egymástól **egy szóközzel** elválasztott **különböző szavakat** tartalmaz. Írjon **programot**, amely feldolgozza az állományt úgy, hogy a szavakból egy **hash függvény** segítségével **1000 elemű táblázatot** állít elő. A hash függvényt tessék úgy megválasztani, hogy az vegye figyelembe a szavak hosszát is! A szinonimákat láncolni kell.

6. Egy rendezett nembináris fát, amelyben minden elemnek legfeljebb K rákövetkezője lehet, reprezentáljunk úgy, hogy minden elem mellé felveszünk egy K elemű mutatóvektort. Írjon **eljárást**, amely úgy járja be a fát, hogy először feldolgozza a **gyökérelemet**, majd **jobbról balra** a rákövetkezőit!

7. Egy operációs rendszerben a memória szabad helyeit **egyirányban láncolt listával** kezeljük, **méret**, **következő szabad hely címe** formában, a **cím** szerint **növekvően**. Készítsen egy olyan **eljárást**, amely paraméterként megkapja a lista **fejét** és egy **tartomány címét és méretét**, és ezt a tartományt felfűzi a listába. Ha a tartomány összeér egy másik tartománnyal (folytonos tárterület!), akkor azokat össze kell olvasztani és a listában csak egyszer kell szerepeltetni őket.

2.12. 1998. május 29., nappali tagozat

1. Adva van egy szövegállomány, amely egy szintaktikusan helyes Pascal programot tartalmaz. A főprogramban egyetlen `const` utasítás van, amelyben **skalár** nevesített konstansokat deklaráltunk.

Írjunk **programot**, amely a program szövegében a nevesített konstansok **neveit kicseréli az értékükre**!

2. Adva van egy szövegállomány, amely **magyar szavakat** tartalmaz. Minden szó **után egy szóköz** áll. A sorokban maximum 30 szó lehet. Egy szó maximum 10 karakterből áll. Írjon **programot**, amely egy másik szövegállományba másolja azokat a **sorokat**, amelyekben **nem** fordul elő a **leggyakoribb** szó (több ilyen is lehet)!

3. Adva van egy szabályos Pascal kifejezés **fája**, az elemek **sztringek**. Írjon egy olyan **függvényt**, amely megkapja a fa **gyökérmutatóját**, és **sztringként** visszaadja a kifejezés **teljesen zárójelezett** alakját!

4. Egy operációs rendszerben a memória szabad helyeit **egyirányban láncolt listával** kezeljük, **méret**, **következő szabad hely címe** formájában, a **cím** szerint **növekvően**. Készítsen egy olyan **eljárást**, amely megkapja a lista **fejét** és egy **méretet**, és megadja annak a szabad helynek a **címét**, amelyik a méretet felülről leginkább megközelíti! Ha több ilyen van, akkor válasszuk a **legnagyobb** címűt! Ha a megtalált szabad hely mérete 2K-nál kevesebbel tér el a megadott mérettől, akkor **töröljük** a listából az őt reprezentáló elemet, egyébként maradjon a listában egy **megfelelően csökkentett** méretű elem!

5. Adva van egy tipizált állomány, amely **kvadratikus ritka mátrixok négysoros reprezentációját** tartalmazza (a mátrix elemei **valósak**, a standard elem a **nulla**). Írjon olyan **logikai függvényt**, amely tetszőleges ilyen állomány esetén eldönti, hogy a mátrix **főátlója alatti** elemek között van-e **nem nulla** elem!

2.13. 1998. június 2., nappali tagozat

1. Adva van egy szövegállomány, amely egy szintaktikusan helyes Pascal programot tartalmaz. Írjon **programot**, amely meghatározza, hogy az **alapszavak** milyen **gyakorisággal** fordulnak elő a programban! Az eredményt egy **szövegállományban** helyezze el!

2. Adva van egy szövegállomány, amely **magyar szavakat** tartalmaz. Minden szó **után egy szóköz** áll. A sorokban maximum 30 szó lehet. Egy szó maximum 10 karakterből áll. Írjon **programot**, amely egy másik **szövegállományba** másolja át azokat a **szavakat** (utánuk egy szóközzel), amelyekben **több** a magánhangzó, mint a mássalhangzó! Az output állományban ne legyenek sorok!

3. Adva van egy szabályos Pascal kifejezés **fája**. Írjon egy olyan **függvényt**, amely megkapja a fa **gyökérmutatóját** és meghatározza, hogy **hány egyoperandusú operátor** van benne!

4. Egy operációs rendszerben a memória szabad helyeit **egyirányban láncolt listával** kezeljük, **méret**, **cím** formában, a **cím** szerint **növekvően**. Készítsen egy olyan **eljárást**, amely paraméterként megkapja a lista **fejét** és egy **méretet**, és meghatározza azon **két** szabad hely **címét**, amelyek mérete **együttesen** a megadott méretet felülről a legjobban közelíti (a megadott méret nagyobb bármely listabeli elem méreténél).

5. Adva vannak **tipizált** állományok, amelyek mindegyike egy-egy **kvadratikus ritka mátrix négysoros reprezentációját** tartalmazza (a mátrix elemei **egészek**, a standard elem a **nulla**), Az állomány első eleme a mátrix méretét tartalmazza **0, 0, méret, 0** formában. Írjon olyan **logikai függvényt**, amely tetszőleges ilyen állomány esetén eldönti, hogy a mátrix **szimmetrikus-e**!

2.14. 1998. december 12., nappali tagozat

1. Adva van egy szövegállomány, amely egy szintaktikusan helyes Pascal programot tartalmaz, amelyben vannak **függvénydeklarációk**. Írjon **eljárást**, amely paraméterként megkapja az **állomány nevét**, és egy **másik** szövegállományba átírja a főprogram **első függvényének** a szövegét!

2. Adva van egy **egészeket** tartalmazó **keresőfa**, és tudjuk, hogy az elemek között van olyan, amelynek értéke megegyezik az elemek **átlagával**. Írjon **függvényt**, amely paraméterként megkapja a fa gyökerét címző mutatót, és meghatározza, hogy a rendezett elemsorozatban **hányadik** a fenti elem!

3. Írjon **programot**, amely egy **billentyűzetről** megadott karaktersorozatról eldönti, hogy az szabályos Pascal kifejezés-e! A kifejezés operandusai **kétkarakteres** azonosítók, illetve háromjegyű előjel nélküli **egész számok** lehetnek. Operátorok: $+$, $-$, $*$, $/$. Zárójelek tetszőlegesen szerepelhetnek. A program írja **képernyőre** a beolvasott karaktersorozatot. Adja a `szabályos kifejezés` üzenetet, ha szabályos, és adjon értelmes hibaüzenetet, ha nem szabályos!

4. Adva van egy **sztringeket** tartalmazó **multilista** két mutatóval. Az egyik lánc a **bekerülés sorrendjében** fűzi fel az elemeket, tehát mindig a végén bővítünk. A másik lánc a sztringeket **hosszuk** szerint **csökkenő** sorrendben tartalmazza. Azonos hosszak esetén szintén a bekerülési sorrendet kell tartani. Írjon **eljárást**, amely **paraméterként** megkapja a két láncfejet és egy sztringet, és a sztringet **beilleszti** a multilistába!

5. Adva van egy szövegállomány, amely soronként egymástól vesszővel elválasztott **egész** értékpárokat tartalmaz. Tekintsük ezeket intervallumoknak. Maximum ezer sor van. Írjon **programot**, amely **képernyőre** írja azt az intervallumot (több ilyen is lehet), amely a **legtöbb** másik intervallumot tartalmazza!

2.15. 1999. május 17., nappali tagozat

1. Írjon egy olyan **függvényt**, amely egy **nemüres**, **nembináris** fa **bináris** reprezentációját állítja elő! A nembináris fában minden elemnek legfeljebb **5** rákövetkezője lehet.

2. Adva van egy C forrásállomány, a forrásszöveg elején paraméter nélküli **define** makrókkal. Írjon **programot**, amely elvégzi a makróhelyettesítést!

3. Írjon **függvényt**, amely egy **tetszőleges bináris fában** megadja azon elemek **darabszámát**, amelyeknek nincs **jobb oldali** rákövetkezőjük!

4. Írja meg azt a **blokkot**, amely kiértékel egy billentyűzetről megadott **prefix** kifejezést! A kifejezésben csak a kétoperandusú $+$, $-$, $*$, $/$ operátorok és előjel nélküli egész számok mint operandusok szerepelnek. Az operátorokat és operandusokat egy-egy **szóköz** választja el egymástól. Ha a kifejezés **szabályos**, akkor a **képernyőn** jelenjen meg az értéke! Ha **nem szabályos**, akkor írja képernyőre a **kifejezést**, és jelölje meg a hiba **helyét** és **okát**!

5. Írjon **programot**, amely egy szöveges állományból leválogatja és képernyőre írja azokat a szavakat, amelyek egy billentyűzetről megadott szó anagrammái (pontosan ugyanazokat a betűket tartalmazzák)! Az állományban a szavakat egy szóköz választja el. A szavak az angol ábécé betűit tartalmazzák. A kis- és nagybetűket nem különböztetjük meg.

2.16. 1999. május 21., esti tagozat

1. Írjon egy **függvényt**, amely **tetszőleges bináris fában** megadja azon elemek **darabszámát**, amelyeknek **két** rákövetkezőjük van!

2. Írja meg azt a **blokkot**, amely kiértékel egy **billentyűzetről** megadott **postfix** kifejezést! A kifejezésben csak a kétoperandusú $+$, $-$, $*$, $/$ operátorok és előjel nélküli egész számok mint operandusok szerepelnek. Az operátorokat és operandusokat egy-egy szóköz választja el egymástól. Ha a kifejezés **szabályos**, akkor a **képernyőn** jelenjen meg az értéke. Ha nem szabályos, akkor írja képernyőre a **kifejezést** és a `Hiba!` üzenetet!

3. Adva van egy **szavakat** tartalmazó szöveges állomány. A szavakban csak az angol ábécé betűi szerepelnek. A szavakat egy-egy szóköz választja el egymástól. Írjon **programot**, amely egy **szövegállományba** írja azokat a szavakat, amelyek **többször** is előfordulnak az állományban!

4. Írjon **függvényt**, amely egy **valósakat** tartalmazó kétdimenziós tömb azon sorainak **indexét** adja meg, amelyekben az elemek **összege** a **legkisebb**!

5. Írjon **programot**, amely egy szöveges állomány azon sorait, amelyek megegyeznek a fordítottjukkal, egy másik szöveges állományba másolja át!

2.17. 1999. december 10., levelező tagozat

1. Adva van egy `szavak.txt` nevű szöveges állomány, amelyben különböző **angol** szavak vannak. Minden szó után **egy szóköz** áll.

   Két szó „távolságán" értsük a következőt: a szavakat karakterről karakterre összehasonlítjuk. Eltérő karakter esetén a résztávolság 1, azonos mássalhangzó esetén 0.5, azonos magánhangzó esetén 0.1. A távolságot a résztávolságok összege adja. Ebből le kell vonni még annyiszor 0.1-et, ahány karakterrel különbözik a két szó hossza.

   Írjon **programot**, amely a **billentyűzetről** beolvas egy **szót**, majd az állomány szavai közül a képernyőre írja azt az **öt** szót, amely a legközelebb van a beolvasott szóhoz! A szavak mellett szerepeljen a távolság is!

2. Írjon egy olyan **függvényt**, amely egy **paraméterként** megadott, **valósakat** tartalmazó kétdimenziós tömb azon **sorának indexét** adja meg (egy ilyen van), amelyben az elemek **összege a legkisebb**!

3. Írjon egy olyan **eljárást**, amely egy **paraméterként** megadott, **egészeket** tartalmazó vektor elemeit úgy rendezi át, hogy elöl álljanak a negatívak, aztán a pozitívak, aztán a nulla értékűek! Az elemek az egymáshoz viszonyított relatív sorrendjüket tartsák meg!

4. Írjon egy **programot**, amely billentyűzetről beolvas egy **egészeket** tartalmazó vektort, majd eldönti, hogy van-e olyan eleme, amelynek értéke **megegyezik** az elemek **átlagával**! Az ilyen elem(ek) indexét írassa ki a képernyőre, vagy ha nincs ilyen elem, akkor egy ilyen értelmű szöveg jelenjen meg!

5. Írjon **programot**, amely egy szöveges állomány azon sorait, amelyek megegyeznek a fordítottjukkal, egy másik szöveges állományba másolja át!

2.18. 2000. január 12., levelező tagozat

1. Írjon egy olyan **függvényt**, amely egy **paraméterként** megadott, **egészeket** tartalmazó kétdimenziós tömb azon **oszlopának** az **indexét** adja meg (csak egyetlen ilyen van), amelyben a **legtöbb pozitív** elem szerepel!

2. Írjon egy olyan **programot**, amely a billentyűzetről **magyar szavakat** olvas be a +++ végjelig, majd egy szövegállományba kiírja a szavakat ábécé sorrendbe rendezve!

3. Írjon **eljárást**, amely egy **paraméterként** megadott szöveges állomány **legrövidebb** sorait (több ilyen is lehet) a képernyőre írja!

4. Írjon egy **eljárást**, amely **paraméterként** egy olyan egydimenziós tömböt kap, amelynek elemei **angol** szavakat tartalmaznak! Az eljárás írja egy **globális** (már létező) szövegállomány **végére** azokat a szavakat, amelyekben a magánhangzók száma megegyezik a mássalhangzók számával!

5. Írjon egy olyan **függvényt**, amely egy **paraméterként** megadott, **különböző egészeket** tartalmazó egydimenziós tömb **második** legkisebb elemét határozza meg!

2.19. 2000. május 10., levelező tagozat

1. Írjon egy olyan **függvényt**, amely egy **paraméterként** megkapott, **egészeket** tartalmazó **kétdimenziós** tömb azon **sorának indexét** adja vissza, amely sorösszeg **nulla** (pontosan egy ilyen sor van)!

2. Írjon **eljárást**, amely egy **paraméterként** megkapott, **karaktereket** tartalmazó **egydimenziós** tömbben meghatározza azon **rész kezdő-** és **végindexét**, amelyben **azonos** karakterek vannak! Több ilyen rész esetén válassza ki azt, amelyben a karakterek száma **maximális**!

3. Írjon **programot**, amely a **billentyűzetről angol** szavakat olvas be a $*$ végjelig, majd **képernyőre** írja azokat a szavakat, amelyekben a **magánhangzók** száma **több**, mint a **mássalhangzók** száma!

4. Adva van egy **szöveges állomány**. Írjon **programot**, amely végigolvassa az állományt, és átmásolja egy **másik** szöveges állományba azokat a sorokat, amelyekben valahol **két szóköz** áll egymás mellett!

5. Írjon egy **függvényt**, amely egy **paraméterként** megkapott, **egészeket** tartalmazó **négyzetes mátrix** esetén a következő értékkel tér vissza:

   1, ha a mátrix szimmetrikus,
   2, ha a mátrix alsó háromszög mátrix,
   3, ha a mátrix felső háromszög mátrix.

2.20. 2000. május 20., nappali tagozat

1. Írjon egy **függvényt**, amely paraméterként megkapott **magyar** szóról eldönti, hogy az **milyen hangrendű**! A visszatérési érték:

   1, ha magas,
   0, ha mély,
   −1, ha vegyes.

2. Adva van egy olyan **szöveges állomány**, amelyben a sorok hossza maximum 80 karakter. Írjon egy olyan **programot**, amely minden sort szóközök segítségével 80 hosszúságúra egészít ki úgy, hogy a szóközöket a szövegelemek között **egyenletesen** osztja szét (sorkizárás)! Az új sorokat helyezze el egy másik szöveges állományban!

3. Adott egy csak **bináris** operátorokat tartalmazó **szabályos** kifejezés fájának **postorder** bejárásával kapott sorozat. Az operátorokat és az operandusokat **egy szóköz** választja el. A sorozatot billentyűzetről kapjuk. Írjon **programot**, amely képernyőre írja az adott kifejezés **preorder** alakját!

4. Írjon egy olyan **eljárást**, amely egy paraméterként megkapott maximum négyjegyű pozitív egész szám értékét **betűvel** a képernyőre írja! (Például **615** esetén **hatszáztizenöt**öt.)

5. Adva van egy teljesen zárójelezett szabályos kifejezés. Írjon egy olyan **függvényt**, amely meghatározza a kifejezés fájának magasságát!

6. Írjon egy olyan **függvényt**, amely egy sztringről eldönti, hogy az egy szabályos C-beli felsorolásos típusú definíciót tartalmaz-e!

2.21. 2000. augusztus 21., esti tagozat

1. Írjon **programot**, amely egy C forrásprogram első utasításaként megadott **nevesített konstans** definíciót figyelembe véve a forrásszövegben a nevet mindenütt helyettesíti a konstanssal!

2. Írjon egy **logikai függvényt**, amely egy paraméterként megkapott **magyar** szóról eldönti, hogy magánhangzóra vagy mássalhangzóra végződik-e!

3. Írjon egy **programot**, amely egy szöveges állomány soraiból eltávolítja a fölösleges (egymás mellett álló) szóközöket!

4. Adott egy csak **bináris** operátorokat tartalmazó **szabályos** kifejezés fájának **postorder** bejárásával kapott sorozat. Az operátorokat és az operandusokat **egy szóköz** választja el. A sorozatot billentyűzetről kapjuk. Írjon **programot**, amely képernyőre írja az adott kifejezés **preorder** alakját!

5. A felvételi után adott a felvételizők eredményeit tartalmazó **egyirányban láncolt lista**, amely elemei a nevet, értesítési címet és az elért pontszámot tartalmazzák. Az adatelemek rendezettek **pontszám** szerint **csökkenőleg**. Adott a biztos felvétel ponthatára és a fellebbezési lehetőség alsó ponthatára. Írjon **eljárást**, amely megadja a fellebbezésre jogosultak adatait, és azt névsor szerint rendezve elhelyezi egy másik egyirányban láncolt listában!

6. Írjon egy **függvényt**, amely **egészeket** tartalmazó **kétdimenziós** tömb azon **oszlopainak** (több ilyen is lehet) az **indexét** adja meg, amelyekben a pozitív elemek száma nagyobb a negatívakénál!

2.22. 2000. december 1., nappali tagozat

1. Adva van egy olyan **szöveges állomány**, amelynek sorai **magyar szavakat** tartalmaznak (legalább egyet és legfeljebb nyolcat), minden szó után **egyetlen szóköz** áll. Írjon **programot**, amely képernyőre írja azokat a **sorokat** (több ilyen is lehet), amelyekben a **legkevesebb** szó van!

2. Írjon egy olyan **logikai függvényt**, amely **igaz** értéket ad, ha a paraméterként megadott, **valós** értékeket tartalmazó egydimenziós tömb **utolsó** elemének értéke megegyezik az elemek átlagának értékével, egyébként pedig **hamis** értékkel tér vissza.

3. Írjon **programot**, amely egy **szöveges állomány** azon sorait, amelyekben a legtöbb **azonos** karakter van, egy másik szöveges állományba írja át!

4. Írjon egy olyan **függvényt**, amely egy paraméterként megkapott, **egészeket** tartalmazó kétdimenziós tömb azon elemeinek **darabszámát** adja meg, ahol az elemek értéke megegyezik az adott elem indexeinek szorzatával (az indexek típusa egész)!

5. Írjon **eljárást**, amely egy első paraméterként megkapott, **sztringeket** tartalmazó egydimenziós tömb minden elemében a második paraméterként megkapott **karakterrel** megegyező karaktereket átírja a harmadik paraméterként megkapott **karakterre**!

2.23. 2001. január 3., levelező tagozat

1. Adva van egy szöveges állomány, amelynek minden sora **vesszővel elválasztott angol szavakat** tartalmaz. A sorokban mindig van legalább **két** szó. Írjon **programot**, amely egy másik szöveges állományba másolja át azokat a sorokat, amelyeknek **utolsó** szava **magánhangzóval** kezdődik!

2. Írjon egy olyan **függvényt**, amely a paraméterként megkapott, **egészeket** tartalmazó **kétdimenziós** tömb azon **oszlopának** az **indexét** adja meg, amely oszlopban a legtöbb **0** elem van! Feltételezzük, hogy csak **egyetlen** ilyen oszlop létezik.

3. Írjon olyan **eljárást**, amely egy paraméterként megkapott **sztringet** megfordít!

4. Adva van egy olyan szöveges állomány, amely egy olyan Pascal (C) programot tartalmaz, amelyik csak **főprogramból** áll. Írjon **függvényt**, amely megadja, hogy a programban hány **változót** deklaráltak!

5. Írjon egy olyan **programot**, amely billentyűzetről **sztringeket** olvas mindaddig, amíg egy üres sztringet nem adunk meg, majd ezekből a sztringekből **hármasával** összeállít egy-egy **sort**, és elhelyezi azt egy **új** szöveges állományban!

2.24. 2001. április 28., nappali tagozat

1. Írjon **programot**, amely egy **létező szöveges állományból**, amelynek soraiban **egyetlen szóközzel elválasztott angol** szavak vannak, egy másik, **még nem létező szöveges állományba** írja át a **magánhangzóra végződő** szavakat! **A sorok száma ne változzon!**

2. Írjon egy **függvényt**, amely egy paraméterként megkapott, **egészeket** tartalmazó **egydimenziós tömbben** megszámolja azon elemek számát, amelyek egy szintén **paraméterként megadott** értéknél **nagyobbak**!

3. Írjon egy **eljárást**, amely a **billentyűzetről beolvas** egy **magyar mondatot**, és a benne szereplő **szavak kezdőbetűjét nagybetűsre** írja át, majd a képernyőre írja a mondatot!

4. Írjon egy **eljárást**, amely egy **paraméterként megkapott sztringben** megkeresi az összes **sorszámot** (pl. **1.**, **2.**, **5.**), és azokat lecseréli a **betűzött** alakjukra (**első**, **második**, **ötödik**)!

5. Írjon egy **eljárást**, amely egy **paraméterként megkapott**, **angol** szavakat tartalmazó egydimenziós tömb elemeit rendezi nagyság szerint **csökkenő** sorrendbe a **végük** szerint!

2.25. 2001. május 14., nappali tagozat

1. Az Elektronikus Tanulmányi Osztály két szöveges állományt használ a nyilvántartásban. Az egyik állomány **soronként** tartalmazza a **tantárgykódot 5** karakteren és a **tantárgy megnevezését maximum 35** karakteren. Az állomány a tantárgykódokat **növekvő** sorrendben tartalmazza. A másik állomány **soronként** tartalmazza a **hallgató nevét pontosan 25** karakteren, a **szakot** és **évfolyamot 4** karakteren és a hallgató által az adott félévben felvett **maximum 7** tantárgy **kódját** és **jegyét**. Írjon **programot**, amely egy szöveges állomány **soraiba** írja azon tantárgyak **nevét**, amelyeket az adott félévben senki sem vett fel! Tudjuk, hogy **77** tantárgy van.

2. Írjon egy **eljárást**, amely egy paraméterként megkapott **sztringben** megkeresi az összes **sorszámot** (pl. **351.**), és azokat lecseréli a betűzött alakjukra (pl. **háromszázötvenegyedik**)! A sorszámok **350** és **399** közé esnek.

3. Írjon egy **logikai függvényt**, amely egy paraméterként megadott, **egészeket** tartalmazó **kétdimenziós** tömb esetén **igaz** értéket ad, ha van a tömbben legalább két olyan **oszlop**, amelynek **átlaga** megegyezik!

4. Adva van egy szöveges állomány, amely egy olyan szabályos **C függvényt** tartalmaz, amelynek **3** formális paramétere van. Írjon **programot**, amely **képernyőre** írja, hogy az egyes formális paraméterekre a függvény mely **soraiban** történik hivatkozás! Feltehetjük, hogy minden sorban egy utasítás áll.

5. Írjon egy **függvényt**, amely paraméterként egy **kifejezés fáját** kapja meg, és visszaadja azt a **sztringet**, amely a kifejezés **teljesen zárójelezett infix** alakját tartalmazza!

2.26. 2001. augusztus 21., esti tagozat

1. Írjon egy **függvényt**, amely egy paraméterként megkapott, **egészeket** tartalmazó **kétdimenziós** tömb esetén megadja azon **oszlopok** számát, amelyekben egy szintén paraméterként megadott értéknél csak **nagyobb** értékek szerepelnek! (1.36.)

2. Írjon egy **programot**, amely egy szöveges állomány sorait úgy írja át egy másik szöveges állományba, hogy megkeresi a **leghosszabb** sort, és az összes többi sort az **elején** kiegészíti annyi **szóközzel**, hogy minden sor hossza azonos legyen!

3. Adva van egy szöveges állomány, amely egy olyan szabályos **C függvényt** tartalmaz, amelynek **egy (1)** formális paramétere van. Írjon egy **logikai függvényt**, amely akkor ad igaz értéket, ha a függvény **rekurzív**! (1.69.)

4. Írjon egy **eljárást**, amely egy paraméterként megkapott, **karaktereket** tartalmazó **bináris fában** a kisbetűket nagybetűre cseréli! (1.48.)

5. Adva van két szöveges állomány. Az egyikben a **tantárgykódok** (**5** karakteren) és a tantárgyak **nevei** (maximum **35** karakteren) vannak. A másikban a hallgatók **neve** (pontosan **25** karakteren) és az általa felvett maximum **12** tantárgy **kódja** szerepel. Írjon **programot**, amely egy szöveges állományba írja azon **tantárgy(ak) nevét**, amelyet a **legtöbben** vettek fel!

2.27. 2001. október 26., levelező tagozat

1. Írjon **programot**, amely egy **magyar** szavakat tartalmazó **szöveges állomány** szavait ábécé sorrendben írja át egy **másik** állományba! (1.66.)

2. Írjon **függvényt**, amely egy paraméterként kapott, **egészeket** tartalmazó **kétdimenziós** tömb azon oszlopának **indexét** adja vissza, amelyben a **legkevesebb pozitív elem** van! (1.35.)

3. Adott egy **szöveges** állomány, amelyben **magyar** szavak vannak, minden szó után egy **szóköz** áll. Írjon **eljárást**, amely képernyőre írja azon sorokat (több ilyen is lehet), amelyekben a **legkevesebb szó** van! (1.60.)

4. Írjon **eljárást**, amely egy paraméterként megkapott, **karaktereket** tartalmazó **egydimenziós** tömbben meghatározza azon rész **kezdő-** és **végindexét**, amelyben **azonos karakterek** vannak! Több ilyen esetén válassza ki azt, amelyben a karakterek száma **maximális**! (1.11.)

5. Írjon **programot**, amely **angol** szavakat kér be **billentyűzetről** *** végjelig, és kiírja egy **szöveges** állományba közülük azokat, amelyek tartalmazzák a `b`, `c`, `x`, `y` karaktereket! (1.57.)

2.28. 2001. november 30., levelező tagozat

1. Írjon egy **függvényt**, amely egy paraméterként megkapott **sztringben szóközzel** felülírja a **nem betű** karaktereket és visszaadja az új sztringet! (1.24.)

2. Írjon **eljárást**, amely paraméterként megkap **két szöveges állomány nevet** és **egy sztringet**, majd az első állomány azon sorait, melyeknek a **vége** azonos a sztringgel, átírja a másik állományba! Az első állomány létezik, a másodikat most kell létrehozni. (1.63.)

3. Írjon egy **programot**, amely a **billentyűzetről** beolvas egy pozitív egész számot (értéke maximum **110** lehet), majd a billentyűzetről beolvas ennyi darab **valós** számot és közülük **képernyőre** írja azokat, amelyek értékének a beolvasott számok **átlagától** való eltérése az átlag felénél nagyobb! (1.9.)

4. Írjon **eljárást**, amely paraméterként megkapott, **tetszőleges méretű**, egészeket tartalmazó kvadratikus mátrixot **tükröz** a **mellékátlójára**! (1.32.)

5. Írjon **eljárást**, amely egy paraméterként kapott **tetszőleges méretű**, egészeket tartalmazó egydimenziós tömbben meghatározza a legnagyobb összegű résztömb **kezdő-** és **végindexét** két output paraméterében! (1.13.)

2.29. 2001. december 17., nappali tagozat

1. Írjon **eljárást**, amely egy paraméterként kapott, **sztringeket** tartalmazó **egydimenziós** tömb minden elemét az **elején** és a **végén egyenletesen elosztott** szóközökkel kiegészíti olyan **hosszúságúra**, mint a **leghosszabb** elem hossza! Az eredeti tömb nem módosulhat! (1.21.)

2. Írjon **függvényt**, amely egy **szintaktikusan helyes C függvényt** tartalmazó **szöveges** állományt dolgoz fel úgy, hogy megszámolja a benne használt **különböző tömböket**!

3. Írjon **programot**, amely **billentyűzetről** beolvas egy **szabályos, teljesen zárójelezett C kifejezést**, amely operandusként csak **konstansokat** és **változókat** tartalmaz, és a **képernyőre** írja azt a **részkifejezést**, amelyet **először** kell kiértékelni! (1.52.)

4. Írjon **eljárást**, amely paraméterként megkap egy **bitmátrixot** és egy **bitvektort**, majd a bitmátrix minden oszlopa és a bitvektor között **kizáró vagy** műveletet végez! Az eredeti bitmátrixra a továbbiakban nincs szükség. (1.33.)

5. Írjon **logikai függvényt**, amely egy paraméterként megkapott, **egészeket** tartalmazó, **szigorú értelemben vett bináris fáról** eldönti, hogy **tökéletesen kiegyensúlyozott-e**! (1.47.)

2.30. 2001. december 18., levelező tagozat

1. Írjon egy **eljárást**, amely egy paraméterként megkapott, **tetszőleges méretű**, egészeket tartalmazó kvadratikus mátrix **főátlójában** elhelyezi **soronként** a főátló fölötti elemek **összegét**! (1.45.)

2. Írjon egy **függvényt**, amelynek első paramétere egy **sztring**, második paramétere egy **pozitív egész szám**, és a függvény adja vissza azt a sztringet, amely az eredetiből úgy keletkezik, hogy azt az elején és a végén kiegészítjük **szóközökkel** (egyenletesen elosztva azokat) úgy, hogy az új sztring hossza a második paraméter értéke legyen! (1.29.)

3. Írjon **programot**, amely egy szöveges állományban elhelyezett, **szintaktikailag helyes** Pascal (C) forrásprogram szövegét úgy másolja át egy másik szöveges állományba, hogy közben kihagyja belőle a **megjegyzéseket**! (1.67.)

4. Írjon egy **eljárást**, amely a paraméterként megkapott, **tetszőleges méretű**, **sztringeket** tartalmazó, egydimenziós tömböt a sztringek **hosszának csökkenő** sorrendjébe teszi! Azonos hosszak esetén a sztringek sorrendje az eredeti sorrend legyen! (1.15.)

5. Írjon **programot**, amely a billentyűzetről pozitív egész számokat olvas mindaddig, míg 0-t nem adunk. A program válassza ki a számok közül a **legkisebbeket** és a **legnagyobbakat** (lehetnek azonos értékűek is, de legfeljebb 3-3), írja azok értékét a **képernyőre**, és adja meg, hogy **hányadikként** olvastuk be őket! (1.4.)

2.31. 2002. január 2., levelező tagozat

1. Írjon **logikai függvényt**, amely egy paraméterként megadott, sztringeket tartalmazó négyzetes mátrixról eldönti, hogy **szimmetrikus-e**! (1.31.)

2. Írjon **eljárást**, amely egy paraméterként megkapott **bitmátrix** minden sora és egy szintén paraméterként kapott **bitvektor** között elvégzi a **vagy** műveletet!

3. Írjon **eljárást**, amely egy paraméterként megkapott, angol szavakat tartalmazó egydimenziós tömbben meghatározza és **képernyőre** írja a szavak **gyakoriságát**! (1.17.)

4. Írjon **programot**, amely billentyűzetről egyenként, elsőre nem meghatározható darabszámú **pozitív egész** értéket olvas be mindaddig, amíg **0** értéket nem kap! A program minden érték beolvasása után határozza meg az **addig beolvasott** értékek **átlagát**! Ha az új érték az eddigi átlag **kétszeresénél** nagyobb, akkor írja képernyőre az eddig beolvasott értékek **darabszámát** és az **új** értéket, majd a hátralévő értékeket másolja át egy **szöveges állományba**, különben viszont a beolvasás végén írja képernyőre a **darabszámot** és az **átlagot**!

5. Írjon **eljárást**, amely egy paraméterként megadott, **sztringeket** tartalmazó egydimenziós tömböt **elemeinek hossza** szerint **csökkenő** sorrendbe rendez! (1.14.)

2.32. 2002. január 2., nappali tagozat

1. Adva van egy olyan **szöveges állomány**, amely sorai **egyetlen szóközzel** elválasztott angol szavakat tartalmaznak. Írjon **programot**, amely meghatározza és **képernyőre** írja a szövegben előforduló szavak **gyakoriságát**! (1.62.)

2. Írjon **programot**, amely **billentyűzetről** beolvas egy olyan **teljesen zárójelezett kifejezést**, mely csak a $+$, $-$, $*$, / operátorokat és olyan **változó** operandusokat tartalmaz, amelyek nevében csak **betű** szerepel! Tudjuk, hogy a kifejezésben **zárójelhiba** van. A program írja képernyőre a kifejezést, és jelölje meg a hiba helyét! (1.55.)

3. Írjon **eljárást**, amely egy paraméterként megkapott, **egészeket** tartalmazó **keresőfából** kitöröl egy szintén paraméterként megadott **értéket**! Az eljárás írjon megfelelő hibaüzenetet a képernyőre, ha a törlés valamilyen okból nem hajtható végre! (1.50.)

4. Írjon **logikai függvényt**, amely egy paraméterként megkapott, **sztringeket** tartalmazó négyzetes mátrixról eldönti, hogy **szimmetrikus-e**! (1.31.)

5. Írjon **függvényt**, amely egy paraméterként megkapott, **szintaktikusan helyes C függvényt** tartalmazó **szöveges** állományt dolgoz fel úgy, hogy meghatározza a függvény által tartalmazott **blokkok** legnagyobb skatulyázási mélységét!

2.33. 2002. április 20., levelező tagozat

1. Írjon **programot**, amely **billentyűzetről** pozitív valós számokat olvas be mindaddig, amíg **nullát** nem adunk (tudjuk, hogy maximum **100** szám lehet). A program írja egy **most létrehozott** szöveges állományba azokat a beolvasott számokat, amelyeknek a **számok átlagától való eltérése** nagyobb, mint az **átlag fele**! (1.10.)

2. Írjon **logikai függvényt**, amely egy paraméterként megkapott **sztringről** eldönti, hogy az **palindróma-e**! (1.22.)

3. Írjon **függvényt**, amely egy paraméterként megkapott **sztringben** az egymás mellett álló **szóközök** közül csak egyet hagy meg, és visszatér az **új** sztringgel! (1.28.)

4. Írjon **eljárást**, amely egy paraméterként megkapott, **egészeket** tartalmazó **kétdimenziós tömb** oszlopait úgy rendezi át, hogy az első sor elemei nagyság szerint **csökkenő sorrendben** legyenek! Feltehetjük, hogy az első sor elemei különbözőek. (1.38.)

5. Írjon **eljárást**, amely egy paraméterként megkapott, **sztringeket** tartalmazó **egydimenziós tömb** minden elemét **azonos hosszúságúra** (a maximális hosszúságú elem hosszúságára) hozza úgy, hogy a sztringek **elejére** megfelelő számú **szóközt** szúr be! (1.19.)

2.34. 2002. május 13., nappali tagozat

1. Írjon **függvényt**, amely egy egydimenziós, sztringeket tartalmazó tömböt kap **paraméterként**, és annak minden elemét **csonkítja** a legrövidebb elem hosszára, és visszaadja az **új** tömböt! (1.20.)

2. Adva van egy **táblázat**, amelynek kulcsa egész, érték része sztring típusú és maximum 500 elem fér el benne. Írjon **függvényt**, amely **paraméterként** megkapja a táblázatot, az aktuális elemszámot és egy egészet. A függvény **bináris kereséssel** keresse meg az adott egész által meghatározott kulcsú elemet és adja vissza annak **érték részét**! Ha nincs ilyen elem, a függvény az **üres sztringgel** térjen vissza! (1.8.)

3. Írjon **eljárást**, amely egy **paraméterként** megadott kétdimenziós, egészeket tartalmazó tömb azon **oszlopát** határozza meg, amelyben benne van az egész tömb **legnagyobb** eleme (csak egy ilyen van)! (1.44.)

4. Adva van egy **szövegállomány**, amely egy olyan **C főprogramot** tartalmaz, amelyben van **felsorolásos típus** deklarálva (több is lehet). Írjon **programot**, amely a főprogram végrehajtható utasításaiban minden felsorolásos típusú literált **felülír** annak értékével, és az új főprogramot elhelyezi egy másik **szöveges** állományban!

5. Írjon **programot**, amely billentyűzetről megkap egy szabályos **prefix** alakú kifejezést. A program írja **képernyőre** az **elsőnek** kiértékelentő részkifejezést **infix** alakban! A kifejezés csak a $+$, $-$, $*$, / kétoperandusú operátorokat és operandusként olyan változókat tartalmaz, amelyek neve egyetlen karakterből áll. (1.53.)

2.35. 2002. május 23., nappali tagozat

1. Adva van egy **szöveges állomány**, amely egymástól egy szóközzel elválasztott **különböző angol** szavakat tartalmaz. Írjon **programot**, amely képernyőre írja azokat a szavakat (ezekből akármennyi lehet), amelyekben a **legtöbb magánhangzó** van. (1.59.)

2. Írjon logikai **függvényt**, amely egy paraméterként kapott **sztringről** eldönti, hogy van-e benne olyan **részsztring**, amely **pontosan 4 azonos** karakterből áll! (1.27.)

3. Írjon **eljárást**, amely egy paraméterként megkapott, **egészeket** tartalmazó, **tetszőleges méretű kétdimenziós** tömb minden olyan elemének a **0** értéket adja, amelynek a **tömb elemeinek átlagától való eltérése az átlag felénél nagyobb**! Az eredeti tömböt változatlanul kell hagyni. (1.46.)

4. Adva van egy **tetszőleges egészeket** tartalmazó **bináris** fa. Írjon **függvényt**, amely paraméterként megkapja a bináris fa **gyökérmutatóját**, a fában elhelyezett értékekből felépít egy **keresőfát**, és visszaadja annak **gyökérmutatóját**! Az új fa elemeinek szerkezete: **kulcs** (a különböző egész értékek), **gyakoriság** (az eredeti fában az adott kulcs hányszor fordult elő). (1.49.)

5. Adva van egy **szöveges** állomány, amely egy **C főprogramot** tartalmaz. Írjon **programot**, amely az összes **azonosítót nagybetűsre** írja át, és az új főprogramot elhelyezi egy **másik szöveges** állományba! (1.68.)

2.36. 2002. augusztus 21., esti tagozat

1. Írjon **logikai függvényt**, amely egy paraméterként megkapott **angol** szó esetén igazzal tér vissza, ha a szóban **nincs egymás mellett két mássalhangzó**! (1.25.)

2. Írjon **programot**, amely egy **létező szöveges állomány** minden sorát **80** karakter hosszúságúra **egészíti ki szóközökkel**, ha rövidebbek a sorok 80 karakternél, és **csonkítja** a végén a sorokat, ha azok hosszabbak 80 karakternél! Az új sorokat egy **új** szöveges állományba kell írni. (1.64.)

3. Írjon **eljárást**, amely egy paraméterként megkapott, **egészeket** tartalmazó **kétdimenziós** tömbben meghatározza azon oszlopok **indexét** (akárhány ilyen lehet), amelyekben a negatív elemek száma **legalább kétszerese** a nulla értékű elemek számának! A tömb mérete **tetszőleges**.(1.43.)

4. Írjon **függvényt**, amely paraméterként egy olyan sztringet kap, amely egy szabályos, **teljesen zárójelezett infix** kifejezést tartalmaz, és meghatározza a kifejezés fájának **magasságát**! A kifejezésben csak a $+$, $-$, $*$, / bináris operátorok és maximum 3 jegyű egész szám operandusok fordulnak elő. (1.51.)

5. Adva van egy **szöveges állomány**, amely soraiban egymástól **egyetlen szóközzel** elválasztott **magyar** szavak állnak. Írjon **eljárást**, amely meghatározza az állományban előforduló szavak **gyakoriságát**! Feltételezhetjük, hogy maximum 200 különböző szó fordul elő. (1.61.)

2.37. 2002. november 29., levelező tagozat

1. Írjon egy **logikai függvényt**, amely egy paraméterként megkapott **sztring** esetén **igaz** értékkel tér vissza, ha a sztringben a betűk (angol ábécé!) száma **nagyobb**, mint a nem-betű karakterek száma, és **hamissal** tér vissza egyébként! (1.23.)

2. Írjon **eljárást**, amely paraméterként megkap egy **tetszőleges** méretű, **egészeket** tartalmazó **egydimenziós** tömböt. Az eljárás határozza meg a tömbben lévő **pozitív**, **negatív** és **nulla** elemek darabszámát! **Az eljárás nem írhat a képernyőre!** (1.6.)

3. Írjon **programot**, amely billentyűzetről egyenként beolvas egész értékeket addig, amíg a -1 értéket nem kapja. A program írja **képernyőre** a beolvasott számok azon szekvenciáját, amely a **leghosszabb szigorúan monoton csökkenő** sorozatot alkotja! (1.12.)

4. Adva van egy **szöveges** állomány, amely egy szintaktikailag helyes, **valós** visszatérési értékkel rendelkező C (Pascal) **függvényt** tartalmaz. Írjon **programot**, amely eldönti, hogy a függvény **rekurzív-e**! Az eredmény jelenjen meg a **képernyőn**! (1.69.)

5. Írjon **függvényt**, amely **tetszőleges** méretű, **valós** értékeket tartalmazó **kétdimenziós** tömböt kap paraméterként. A függvény határozza meg azon **oszlop indexét**, amelyben van olyan elem, amelynek az értéke megegyezik az oszlop elemeinek **átlagával**! Ha több ilyen oszlop is van, akkor a **legnagyobb** indexértéket adja vissza! (1.34.)

2.38. 2003. január 6., esti tagozat

1. Írjon **eljárást**, amely egy paraméterként megkapott, tetszőleges méretű, valósakat tartalmazó kétdimenziós tömb **sorait** úgy rendezi át, hogy az **utolsó oszlop** értékei **csökkenő** sorrendben legyenek! (1.39.)

2. Írjon **logikai függvényt**, amely egy paraméterként megkapott sztringnél igaz értéket ad vissza, ha a sztringben van olyan **4 elemű** részsztring, amely **legalább háromszor** ismétlődik. (1.26.)

3. Adott egy csak a `+`, `-`, `/`, `*` **bináris** operátorokat tartalmazó **szabályos** kifejezés fájának **post-order** bejárásával kapott sorozat. Az operátorokat és az operandusokat **egy szóköz** választja el egymástól. A sorozatot billentyűzetről kapjuk. Írjon **programot**, amely képernyőre írja a kifejezés **prefix** alakját! (1.54.)

4. Írjon **eljárást**, amely egy paraméterként megkapott, tetszőleges méretű, egészeket tartalmazó kétdimenziós tömb **oszlopátlagai** közül meghatározza a **legnagyobbat** (több ilyen is lehet)! **Az eljárás nem írhat képernyőre és állományba!** (1.40.)

5. Adva van egy szöveges állomány, amely egy szintaktikailag helyes, **egész** visszatérési értékkel rendelkező C **függvényt** tartalmaz. Írjon **programot**, amely meghatározza, hogy a függvény **rekurzív-e**! Az eredmény a **képernyőn** jelenjen meg! (1.69.)

2.39. 2003. január 6., levelező tagozat

1. Írjon **eljárást**, amely egy paraméterként megkapott, tetszőleges méretű, valósakat tartalmazó kétdimenziós tömb **sorait** úgy rendezi át, hogy az **utolsó oszlop** értékei **csökkenő** sorrendben legyenek! (1.39.)

2. Írjon **logikai függvényt**, amely egy paraméterként megkapott sztringnél igaz értéket ad vissza, ha a sztring **tükörszimmetrikus** (pl. `görög`, `kosarasok`)! (1.22.)

3. Írjon **programot**, amely 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**! (1.1.)

4. Írjon **eljárást**, amely egy paraméterként megkapott, tetszőleges méretű, egészeket tartalmazó kétdimenziós tömb **oszlopátlagai** közül meghatározza a **legnagyobbat** (több ilyen is lehet)! **Az eljárás nem írhat képernyőre és állományba!** (1.40.)

5. Adva van egy szöveges állomány, amely egy szintaktikailag helyes, **egész** visszatérési értékkel rendelkező C (Pascal) **függvényt** tartalmaz. Írjon **programot**, amely meghatározza, hogy a függvény **rekurzív-e**! Az eredmény a **képernyőn** jelenjen meg! (1.69.)

2.40. 2003. május 17., levelező tagozat

1. Adva van egy **szöveges** állomány, benne egy szintaktikailag szabályos C (Pascal) **függvény**. Írjon **függvényt**, amely megadja a függvény **végrehajtható utasításainak** darabszámát.

2. Írjon **eljárást**, amely paraméterként megkap egy **karaktereket** tartalmazó, **tetszőleges méretű egydimenziós** tömböt, és a tömb **nem betű** karaktereit kicseréli **szóközre**. (1.5.)

3. Írjon **függvényt**, amely paraméterként megkap egy **tetszőleges méretű**, **egészeket** tartalmazó **kvadratikus mátrixot**, és visszaadja a **főátló** maximális és minimális elemét. **A képernyőre nem írunk!** (1.41.)

4. Írjon **eljárást**, amely paraméterként megkap egy **azonos hosszúságú sztringeket** tartalmazó, **tetszőleges méretű egydimenziós** tömböt, továbbá egy **karaktert** és egy **pozitív egész** számot, és a **képernyőre írja** azokat a tömbelemeket, amelyekben a karakter pontosan az adott számszor fordul elő. **A szélsőséges eseteket vizsgálni kell**! (1.16.)

5. Írjon **programot**, amely a **billentyűzetről angol szavakat** olvas mindaddig, amíg **üres sztringet** nem kap. A program írja egy **szöveges állományba** azokat a szavakat, amelyekben egymás mellett van **legalább három mássalhangzó**. (1.58.)

2.41. 2003. május 23., nappali tagozat

1. Adva van egy **szöveges állomány** és benne egy **szintaktikailag szabályos C függvény**. Írjon **eljárást**, amely egy **új** szöveges állományba átmásolja a **formázott** forrásszöveget. A formázás: minden utasítás külön sorban legyen; a blokk kezdő és záró kapcsos zárójele külön sorban legyen; a beágyazott programegységek a tartalmazó programegység utasításaitól 3 karakternyi hellyel beljebb kezdődjenek; a szöveg balra zárt legyen.

2. Írjon **programot**, amely **billentyűzetről** megkap egy olyan **teljesen zárójelezett** kifejezést, amely csak a $-$ és a $+$ **egy-** és **kétoperandusú** operátorokat, operandusként pedig olyan **C-beli változókat** tartalmaz, melyek neve **maximum két** karakterből áll. Ellenőrizze le, hogy a kifejezés **szabályos-e**. A képernyőre írjon értelemszerű hibaüzeneteket. (1.56.)

3. Írjon **függvényt**, amely paraméterként megkap egy olyan **tetszőleges** méretű **egydimenziós** tömböt, amely **angol** szavakat tartalmaz. A függvény **visszatérési értéke** adja meg a szavak **gyakoriságát**. (1.18.)

4. Írjon **eljárást**, amely paraméterként megkap egy **tetszőleges** méretű, **valósakat** tartalmazó **kétdimenziós** tömböt, és előállít egy olyan **egydimenziós** tömböt, amely a **sorok átlagát** tartalmazza. **Az eljárás a képernyőre nem írhat!** (1.37.)

5. Írjon **programot**, amely nullától különböző **egész** értékeket olvas be a **billentyűzetről** a **0** végjelig. A program határozza meg és írja **képernyőre** azt a **három** értéket, amelynek **átlaga maximális**. (1.3.)

2.42. 2003. május 28., nappali tagozat

1. Adva van egy **szöveges állomány** és benne egy **szintaktikailag szabályos C függvény**. Írjon **eljárást**, amely egy **új** szöveges állományba átmásolja a **formázott** forrásszöveget. A formázás: minden utasítás külön sorban legyen; a blokk kezdő és záró kapcsos zárójele külön sorban legyen; a beágyazott programegységek a tartalmazó programegység utasításaitól 3 karakternyi hellyel beljebb kezdődjenek; a szöveg balra zárt legyen.

2. Írjon **függvényt**, amely **paraméterként** megkap egy olyan **teljesen zárójelezett** kifejezést, amely csak a $-$ és a $+$ **egy-** és **kétoperandusú** operátorokat, operandusként pedig **C-beli literálokat** tartalmaz. A kifejezés szintaktikailag **helyes**. A függvény építse fel a kifejezés **fáját**.

3. Írjon **függvényt**, amely paraméterként megkap egy olyan **tetszőleges** méretű **egydimenziós** tömböt, amely **angol** szavakat tartalmaz. A függvény **visszatérési értéke** adja meg azon szavakat, melyekben **legalább 3 mássalhangzó** szerepel egymás mellett.

4. Írjon **eljárást**, amely paraméterként megkap egy **tetszőleges** méretű, **valósakat** tartalmazó **kétdimenziós** tömböt és egy **pozitív valós** értéket, majd meghatározza azon **sorok indexét** (ezekből akármennyi lehet), amelyekben a főátlóbeli elemek sorátlagtól való **eltérése** a második paraméter értékétől **kisebb**. **Az eljárás a képernyőre nem írhat!**

5. Írjon **programot**, amely nullától különböző **egész** értékeket olvas be a **billentyűzetről** a **0** végjelig. A program határozza meg és írja **képernyőre** azt a **leghosszabb** számsorozatot (amennyiben több ilyen is van, akkor mindet), amelyben a számok **előjele azonos**.

2.43. 2003. augusztus 25., esti tagozat

1. Adva van egy szöveges állomány, amely egy **szintaktikailag helyes**, **karakteres** visszatérési értékkel rendelkező **C függvényt** és egy **szintaktikailag helyes főprogramot** tartalmaz. Írjon **programot**, amely megállapítja, hogy a főprogram **szabályosan** hívta-e meg a függvényt!

2. Írjon **eljárást**, amely paraméterként megkap egy **sztringeket** tartalmazó, **tetszőleges** méretű **egydimenziós** tömböt, és ebből a **hívás helyén felhasználható** új egydimenziós tömböt állít elő, amely azokat a **sztringeket** tartalmazza, amelyekben a **betűk** (angol ábécét tételezve fel) száma megegyezik a **nem betű karakterek** darabszámával.

3. Írjon **függvényt**, amely egy paraméterként megkapott, **tetszőleges méretű**, **valósakat** tartalmazó **kétdimenziós** tömb **minimális** és **maximális** átlagú **oszlopainak** (ezekből csak egy-egy lehet) **indexét** határozza meg visszatérési értékként.

4. Írjon **programot**, amely **nullától** különböző **egész** értékeket olvas be a billentyűzetről a **0** végjelig. A program határozza meg és írja **képernyőre** a beolvasott különböző egész **értékeket** és azok **gyakoriságát**.

5. Írjon **függvényt**, amely meghatározza a paramétereként megkapott, **tetszőleges méretű** (de legalább négyelemű), **egészeket** tartalmazó **egydimenziós** tömb **harmadik legkisebb** elemének az **indexét** (amennyiben több ilyen elem is van, akkor a legnagyobb indexet kell visszaadni).

2.44. 2003. december 5., levelező tagozat

1. Írjon **programot**, amely egy **létező** szöveges állomány maximum 100 karakrterből álló sorainak mindegyikét **100 hosszúságúra** egészíti ki, a sorok elején és végén egyenletesen elosztott módon **szóközöket** szúrva be („**középre igazítás**"). Az új sorokat egy **most létrehozott** szöveges állományba kell elhelyezni.

2. Írjon **logikai függvényt**, amely akkor tér vissza igaz értékkel, ha a paraméterként kapott **sztring szabályos C azonosító**.

3. Írjon **eljárást**, amely paraméterként megkap egy **tetszőleges méretű**, **valósakat** tartalmazó **kétdimenziós** tömböt, és meghatározza azt a **vektort**, amely az **oszlopok átlagát** tartalmazza. **Az eljárásban nem lehet semmilyen I/O művelet!**

4. Írjon **függvényt**, amely paraméterként megkap egy **tetszőleges** méretű **négyzetes bitmátrixot** és egy megfelelő méretű **bitvektort**, és visszatér azzal a **bitvektorral**, amely a mátrix főátlója és a vektor elemei között képzett **kizáró vagy** eredményeként keletkezik.

5. Írjon **programot**, amely **billentyűzetről** egész értékeket olvas be a **0** végjelig. A program írja **képernyőre** azokat az értékeket, amelyek **előjele** eltér a **megelőző** érték előjelétől. Például 1 2 3 4 0 esetén nincs ilyen érték, 1 −1 2 −2 −5 0 esetén kiírandó −1 2 −2.

2.45. 2004. január 9., levelező tagozat

1. Írjon **programot**, amely egy **létező** szöveges állomány sorainak mindegyikét **100 hosszúságúra** egészíti ki a sorok végén **szóközöket** szúrva be, ha rövidebb, illetve **elhagyva** a fölösleges karaktereket, ha hosszabb. Az új sorokat egy **most létrehozott** szöveges állományba kell elhelyezni.

2. Írjon **logikai függvényt**, amely akkor tér vissza igaz értékkel, ha a paraméterként kapott **sztring szabványos C egész literál**.

3. Írjon **függvényt**, amely paraméterként megkap egy **tetszőleges méretű**, **valósakat** tartalmazó **kétdimenziós** tömböt, és visszatérési értékként meghatározza a **sorok átlagának minimumát** és az **oszlopok átlagának maximumát**.

4. Írjon **eljárást**, amely paraméterként megkap egy **tetszőleges** méretű, **sztringeket** tartalmazó **vektort**, és előállít egy olyan **vektort**, amelymek elemei rendre a paraméterként kapott vektor elemeinek annyiadik **karakterét** tartalmazzák, amennyi az adott elem **indexe**, illetve a `@` karaktert, ha nem létezik ilyen elem. Egy sztring karaktereit 0-tól sorszámozzuk. **Az eljárásban nem lehet semmilyen I/O művelet!**

5. Írjon **programot**, amely **billentyűzetről** egész értékeket olvas be a **0** végjelig. A program írja **képernyőre** azokat az értékeket, amelyek **megegyeznek** az **előző két** érték **összegével**.

2.46. 2004. május 15., levelező tagozat

1. Írjon **programot**, amely **billentyűzetről nempozitív** valós számokat olvas be mindaddig, amíg **1.1**-et nem adunk. A program írja egy most létrehozott **szöveges** állományba a beolvasott számok közül a **10** legnagyobbat, vagy ha tíznél kevesebb számot olvastunk be, akkor mindet!

2. Írjon **logikai függvényt**, amely egy paraméterként megkapott **sztringről** eldönti, hogy abban egy **másik paraméterként** megkapott **karakter legalább háromszor** előfordul-e!

3. Írjon **függvényt**, amely paraméterként megkapott **sztringben**, amely **szóközökkel** elválasztott tetszőleges karaktersorozatokat tartalmaz, a szóközök és szóközcsoportok helyett **vesszőt** helyez el, és **visszatér az új sztringgel**! Például `"b n mm. 76("` esetén az eredmény `"b,n,mm.,76(".`

4. Írjon **eljárást**, amely egy **paraméterként** megkapott, **egészeket** tartalmazó, tetszőleges méretű, **kétdimenziós** tömb **oszlopai** közül meghatározza azt az oszlopot (mint **egydimenziós** tömböt), amelyben a legtöbb **páros szám** van!

5. Írjon **eljárást**, amely egy **paraméterként** megkapott, **sztringeket** tartalmazó, tetszőleges méretű **egydimenziós** tömb minden elemét egy **másik paraméter** által megadott hosszúságúra **csonkítja**, ha az adott elem **hosszabb**, illetve az **elején szóközökkel** kiegészítve adott hosszúságúra hozza, ha az adott elem **rövidebb**! **Az eredeti tömb nem változhat meg!**

2.47. 2004. június 2., levelező tagozat

1. Írjon **programot**, amely **billentyűzetről egész** számokat olvas be **egyenként** mindaddig, amíg **0**-t nem adunk. A program írja egy most létrehozott **szöveges** állományba **soronként** a beolvasott számok közül azokat az **egymás melletti számhármasokat**, amelyek lehetnek egy **háromszög** oldalhosszai!

2. Írjon **logikai függvényt**, amely egy paraméterként megkapott **sztringről** eldönti, hogy abban **két másik**, szintén **paraméterként** megkapott **karakter** előfordul-e úgy, hogy közöttük pontosan egy szóköz van!

3. Írjon **függvényt**, amely egy **paraméterként** megkapott **tetszőleges sztringben** megszámolja, hogy az **utolsó** karakter hányszor fordul elő!

4. Írjon **eljárást**, amely egy **paraméterként** megkapott, **egészeket** tartalmazó, tetszőleges méretű, **kétdimenziós** tömb oszlopaiból kiválogatja a **legkisebb abszolút értékű** elemeket (oszloponként csak egy-egy ilyen van) és azokat elhelyezi egy **egydimenziós** tömbben!

5. Írjon **függvényt**, amely egy **paraméterként** megkapott, **valósakat** tartalmazó, tetszőleges méretű, **kétdimenziós** tömb **legnagyobb** elemének az **indexeivel** tér vissza!

2.48. 2004. december 3., levelező tagozat

1. Írjon **programot**, amely **billentyűzetről egész** számokat olvas be **egyenként** mindaddig, amíg **0**-t nem adunk. A program írja egy most létrehozott **szöveges** állományba **soronként** a beolvasott számok közül azokat az **egymás melletti legalább 3 elemből álló számsorozatokat**, amelyek számtani sorozatot alkotnak!

2. Írjon **logikai függvényt**, amely egy paraméterként megkapott **sztringről** eldönti, hogy abban **két másik**, szintén **paraméterként** megkapott **karakter** a **megadás sorrendjében** előfordul-e úgy, hogy közöttük **legfeljebb** két másik karakter van!

3. Írjon **eljárást**, amely egy paraméterként megkapott sztringet alakít át. A sztringben olyan mondatok vannak, amelyek végén **.**, **?** vagy **!** áll. A mondatok első karakterét alakítsa nagybetűssé, a többit kisbetűssé!

4. Írjon **eljárást**, amely egy **paraméterként** megkapott, **valósakat** tartalmazó, tetszőleges méretű, **kétdimenziós** tömb minden elemét egészre kerekíti!

5. Írjon **függvényt**, amely egy **paraméterként** megkapott, **egészeket** tartalmazó, tetszőleges méretű **kétdimenziós** tömb **legnagyobb abszolút értékű** elemének (egy ilyen van) az **indexeivel** tér vissza!

2.49. 2004. december 20., levelező tagozat

1. Írjon **programot**, amely **billentyűzetről egész** számokat olvas be **egyenként** mindaddig, amíg **0**-t nem adunk. A program írja egy most létrehozott **szöveges** állományba **soronként** a beolvasott számok közül azokat az **egymás melletti legalább 3 elemből álló** számsorozatokat, amelyek **szigorúan monoton** sorozatot alkotnak!

2. Írjon **logikai függvényt**, amely egy paraméterként megkapott **sztringről** eldönti, hogy abban **két másik**, szintén **paraméterként** megkapott **karakter** előfordul-e egymás mellett úgy, hogy az elsőből pontosan kettő, a másodikból pedig egy szerepel!

3. Írjon **eljárást**, amely egy paraméterként megkapott sztringet úgy alakít át, hogy a benne lévő, **vesszővel** elválasztott szavakat **megfordítja**!

4. Írjon **eljárást**, amely egy **paraméterként** megkapott, **valósakat** tartalmazó, tetszőleges méretű, **kétdimenziós** tömb negatív elemeiből egy vektort készít!

5. Írjon **függvényt**, amely egy **paraméterként** megkapott, **egészeket** tartalmazó, tetszőleges méretű **kétdimenziós** tömb **legnagyobb abszolút értékű** elemének (egy ilyen van) az **indexeivel** tér vissza!

2.50. 2005. május 14., levelező tagozat

1. Írjon **programot**, amely **billentyűzetről egész** számokat olvas be **egyenként** mindaddig, amíg **0**-t nem adunk. A program írja egy most létrehozott **szöveges** állományba **soronként** a beolvasott számok közül azokat az **egymás melletti számhármasokat**, amelyek közül a középső a másik kettő szorzata!

2. Írjon **logikai függvényt**, amely egy paraméterként megkapott **sztringről** eldönti, hogy abban **két másik**, szintén **paraméterként** megkapott **karakter együttesen** egy negyedik paraméter által meghatározott értéknél **többször** fordul elő!

3. Írjon **eljárást**, amely egy paraméterként megkapott sztringet alakít át. A sztringben olyan mondatok vannak, amelyek végén **.**, **?** vagy **!** áll, bennük minden szót **valahány szóköz** választ el. A szavak **első** és **utolsó** karakterét alakítsa **nagybetűssé**, a többit **kisbetűssé**! Az eredeti sztring nem változhat meg!

4. Írjon **eljárást**, amely egy **paraméterként** megkapott, **valósakat** tartalmazó, **tetszőleges** méretű, **kétdimenziós** tömb **oszlopainak átlagát** határozza meg!

5. Írjon **függvényt**, amely egy **paraméterként** megkapott, **sztringeket** tartalmazó, **tetszőleges** méretű **egydimenziós** tömb **leghosszabb** elemeinek (akárhány ilyen lehet) az **indexeivel** tér vissza!

2.51. 2005. június 14., levelező tagozat

1. Írjon **programot**, amely **billentyűzetről negatív egész** számokat olvas be **egyenként** mindaddig, amíg **1**-et nem adunk. A program írja egy most létrehozott **szöveges** állományba **soronként** a beolvasott számok közül azokat a **részsorozatokat**, amelyek **legalább kételeműek** és **szigorúan monoton növekvőek**!

2. Írjon **logikai függvényt**, amely **igennel** tér vissza, ha a paramétereként megkapott **angol szóban** minden betű különbözik!

3. Írjon **függvényt**, amely egy **paraméterként** megkapott **tetszőleges sztringben** megszámolja a karakterek gyakoriságát!

4. Írjon **eljárást**, amely egy **paraméterként** megkapott, valósakat tartalmazó, tetszőleges méretű, **kétdimenziós** tömbből kiválogatja azokat az elemeket, amelyek egy második paraméterként megadott értéknél kevésbé térnek el a tömb összes elemének átlagától, és azokat elhelyezi egy **egydimenziós** tömbben!

5. Írjon **függvényt**, amely egy **paraméterként** megkapott, **egészeket** tartalmazó, tetszőleges méretű **kétdimenziós** tömbben meghatározza, hogy melyik **sorban** (több ilyen is lehet, egy biztos van) fordul elő egy második paraméterként megadott elem, és visszaadja a sorok **indexeit**!