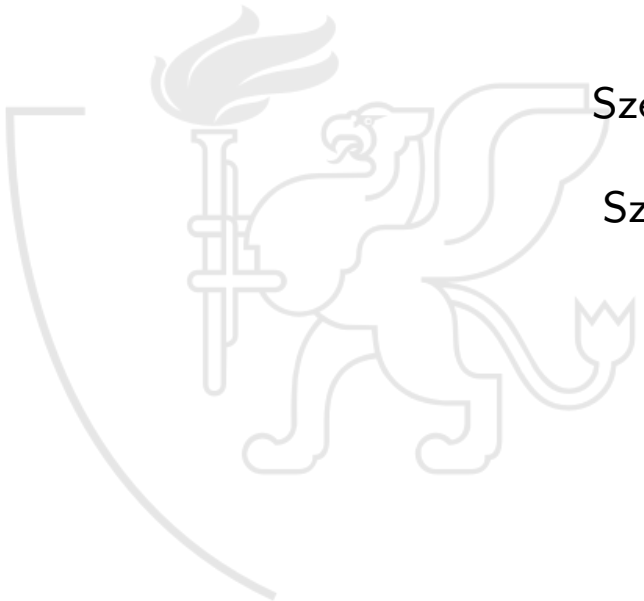
Programozás Alapjai

Előadásprogramok

Dr. Gergely Tamás
Dr. Jász Judit

Szegedi Tudományegyetem
Informatikai Intézet
Szoftverfejlesztés Tanszék



2021

(v0901)

ABC [1/1]

abc.c [1–17]

```c
/* Az ábécé kis- és nagybetűinek kiíratása összefésülve.
 * A for ciklusban a , műveletet alkalmazzuk.
 * 1997. November 7.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    char cha;                       /* az ábécé kisbetűinek */
    char chA;                      /* az ábécé nagybetűinek */

    for (cha = 'a', chA = 'A'; cha <= 'z'; ++cha, ++chA) {
        printf("%c%c", cha, chA);
    }
    printf("\n");
    return 0;
}
```

Angol ABC betűinek összefésült kiíratása

abc.c (v1.0) [1–15]

```c
/* Írassuk ki az angol ábécé kis- és nagybetűit összefésülve.
 *
 * 1997. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    for (char cha = 'a', chA = 'A'; cha <= 'z'; ++cha, ++chA) {
        printf("%c%c", cha, chA);
    }
    printf("\n");
    return 0;
}
```

Parancssori argumentumok kezelése [1/1]

arg.c [1–21]

```c
/* Kiírjuk a parancssorban lévő argumentumokat.
 * 1998. Április 16. Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main(int argc, char **argv) {

/* vagy így is lehet az argv-t deklarálni

int main(int argc, char *argv[]) {

*/
    int i;

    printf("argc = %d\n\n", argc);
    for (i = 0; i < argc; ++i) {
        printf("argv[%d]: %s\n", i, argv[i]);
    }
    return 0;
}
```

Egy útvonal leírásból a fájlnév kinyerése [1/1]

basename.c [1–26]

```c
/* Egy útvonalból a fájlnév kinyerése.
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 1024

int main() {
    char path[N], base[N];
    int i, lastsep = -1;
    fgets(path, N, stdin);
    for (i = 0; path[i] != 0; ++i) {
        if (path[i] == '/') {
            lastsep = i;
        }
    }
    ++lastsep;
    i = 0;
    while (path[lastsep] != 0) {
        base[i++] = path[lastsep++];
    }
    base[i] = 0;
    fputs(base, stdout);
    return 0;
}
```

Basename standard inputról

basename.c (v1.0) [1–25]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 32

void basename(char path[], char base[]) {
    int i = 0, lastsep = -1;
    for (; path[i] != 0; ++i) {
        if (path[i] == '/') {
            lastsep = i;
        }
    }
    ++lastsep;
    i = 0;
    while (path[lastsep] != 0) {
        base[i++] = path[lastsep++];
    }
    base[i] = 0;
}
```

Basename standard inputról

basename.c (v1.0) [27–33]

```
27 int main() {
28     char path[N], base[N];
29     fgets(path, N, stdin);
30     basename(path, base);
31     fputs(base, stdout);
32     return 0;
33 }
```

Basename parancssorból

basename.c (v2.0) [1–26]

```c
/* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
 * a fájl nevét (az útvonal nélkül).
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

const char * basename(const char *path) {
    const char *base = path;
    for (; *path != 0; ++path) {
        if (*path == '/') {
            base = path + 1;
        }
    }
    return base;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        return 1;
    }
    puts(basename(argv[1]));
    return 0;
}
```

Rendezés több szempont szerint [1/8]

beszuro.c [1–25]

```c
/* Rendezzük névsorba illetve átlag szerint a hallgatókat!
 * Flexibilis tömbbel történik a megvalósítás, tehát a
 * névsor hosszát nem kell előre megmondani.
 * 1998. Február 16.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2006. Augusztus 15.  Gergely Tamás, gertom@inf.u-szeged.hu
 * 2014. Október 15.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

#define L 10                                        /* lapméret */

typedef struct {                                  /* a tömb elemtípusa */
    char nev[21];
    float adat;
} elem_t;

typedef elem_t *lap_t;

typedef lap_t  *lapterkep_t;

typedef unsigned int index_t;
```

Rendezés több szempont szerint [2/8]

beszuro.c [27–50]

```
27 typedef struct flex_tomb_t {
28     lapterkep_t lt;                                    /* laptérkép */
29     index_t     hatar;                          /* aktuális indexhatár */
30 } flex_tomb_t;
31
32 typedef bool (*rend_rel_t)(elem_t, elem_t);
33
34 /* A műveletek megvalósítása: */
35
36 void kiolvas(flex_tomb_t a, index_t i, elem_t *x) {
37     if (i < a.hatar) {
38         *x = a.lt[i / L][i % L];
39     }
40 }
41
42 void modosit(flex_tomb_t a, index_t i, elem_t x) {
43     if (i < a.hatar) {
44         a.lt[i / L][i % L] = x;
45     }
46 }
47
48 index_t felso(flex_tomb_t a) {
49     return a.hatar;
50 }
```

Rendezés több szempont szerint [3/8]

beszuro.c [52–75]

```
52 void letesit(flex_tomb_t *a, unsigned int n) {
53     a->hatar = n;
54     if (n) {
55         int j;
56         a->lt = (elem_t**) malloc((1 + ((n - 1) / L)) * sizeof(lap_t));
57         for (j = 0; j <= ((n - 1) / L); ++j) {                /* lapok létesítése */
58             a->lt[j] = (elem_t*) malloc(L * sizeof(elem_t));
59         }
60     } else {
61         a->lt = NULL;
62     }
63 }
64
65 void megszuntet(flex_tomb_t *a) {
66     if (a->hatar) {
67         int j;
68         for (j = 0; j <= ((a->hatar - 1) / L); ++j) {            /* lapok törlése */
69             free(a->lt[j]);
70         }
71         free(a->lt);
72         a->lt = NULL;
73         a->hatar = 0;
74     }
75 }
```

Rendezés több szempont szerint [4/8]

beszuro.c [77–98]

```
77 void novel(flex_tomb_t *a, index_t d) {
78     int j;
79     a->lt = (elem_t**) realloc(a->lt,
80                                (1 + ((a->hatar + d - 1) / L)) * sizeof(lap_t));
81     for (j = ((a->hatar) ? ((a->hatar - 1) / L) + 1 : 0);           /*   új lapok */
82                                j <= (a->hatar + d - 1) / L; ++j) {     /* létesítése */
83         a->lt[j] = (elem_t*) malloc(L * sizeof(elem_t));
84     }
85     a->hatar += d;
86 }
87
88 void csokkent(flex_tomb_t *a, index_t d) {
89     if (d <= a->hatar) {
90         int j;
91         for (j = (a->hatar - d - 1) / L + 1; j <= (a->hatar - 1) / L; ++j) {
92             free(a->lt[j]);                              /* felesleges lapok törlése */
93         }
94         a->hatar -= d;
95         a->lt = (elem_t**) realloc(a->lt,
96                                (1 + ((a->hatar - 1) / L)) * sizeof(lap_t));
97     }
98 }
```

Rendezés több szempont szerint [5/8]

beszuro.c [100–122]

```
100 /* Rendezés */
101
102 void beszuro_rendezes(flex_tomb_t t, rend_rel_t kisebb) {
103                             /* A kisebb rendezési reláció szerinti helyben rendezés */
104     int     i, j;
105     elem_t e, f;
106     for (i = 1; i < felso(t); ++i) {
107         kiolvas(t, i, &e);
108         j = i - 1;
109
110         while (true) {
111             if (j < 0) {
112                 break;
113             }
114             kiolvas(t, j, &f);
115             if (kisebb(f, e)) {
116                 break;
117             }
118             modosit(t, ((j--) + 1), f);
119         }
120         modosit(t, j + 1, e);
121     }
122 } /* beszuro_rendezes */
```

Rendezés több szempont szerint [6/8]

beszuro.c [124–147]

```
124 bool rend_nev_novekvo(elem_t x, elem_t y) {
125                                          /* a névsor szerinti rendezési reláció  */
126     return strcmp(x.nev, y.nev) <= 0;
127 }
128
129 bool rend_adat_novekvo(elem_t x, elem_t y) {
130                                           /* az adat szerinti rendezési reláció  */
131     return x.adat <= y.adat;
132 }
133
134 bool rend_adat_csokkeno(elem_t x, elem_t y) {
135                                        /* az adat szerint csökkenő rendezési reláció  */
136     return x.adat >= y.adat;
137 }
138
139 void kiiras(flex_tomb_t t) {
140                                                                  /* Kiíratás */
141     elem_t e;
142     index_t i;
143     for (i = 0; i < felso(t); ++i) {
144         kiolvas(t, i, &e);
145         printf("%6.2f␣%s\n", e.adat, e.nev);
146     }
147 }
```

Rendezés több szempont szerint [7/8]

beszuro.c [149–165]

```
149 int main() {
150     flex_tomb_t sor;
151     elem_t      hallg;                                  /* beolvasáshoz */
152     index_t     i;
153
154     letesit(&sor, 0);                           /* a flexibilis tömb létesítése */
155                                                         /* beolvasás */
156     printf("Kérem az adatsort, külön sorban név és adat!\n");
157     printf("A végét a *jelzi.\n");
158     scanf("%20[^\n]%*[^\n]", hallg.nev); getchar();
159     i = 0;                                          /* az i. helyre fogunk beírni */
160     while (strcmp(hallg.nev, "*")) {
161         novel(&sor, 1);                             /* a flexibilis tömb bővítése */
162         scanf("%f%*[^\n]", &hallg.adat); getchar();
163         modosit(sor, i++, hallg);
164         scanf("%20[^\n]%*[^\n]", hallg.nev); getchar();
165     }
```

Rendezés több szempont szerint [8/8]

beszuro.c [167–181]

```
167     beszuro_rendezes(sor, rend_nev_novekvo);                /* Rend. névsor szerint */
168     printf("Névsor␣szerint␣rendezve:\n");
169     kiiras(sor);                                                        /* Kiíratás */
170
171     beszuro_rendezes(sor, rend_adat_novekvo);                 /* Rend. adat szerint */
172     printf("Adat␣szerint␣rendezve:\n");
173     kiiras(sor);                                                        /* Kiíratás */
174
175     beszuro_rendezes(sor, rend_adat_csokkeno);                      /* Rendezés újra */
176     printf("Adat␣szerint␣csökkenő␣sorba␣rendezve:\n");
177     kiiras(sor);                                                        /* Kiíratás */
178
179     megszuntet(&sor);                                  /* a flexibilis tömb törlése */
180     return 0;
181 }
```

Adatok rendezése

beszuro.c (v1.0) [1–27]

```c
/* Rendezzük névsorba illetve átlag szerint a hallgatókat!
 * Flexibilis tömbbel történik a megvalósítás, tehát a
 * névsor hosszát nem kell előre megmondani.
 *
 * 1998. Február 16. Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2006. Augusztus 15. Gergely Tamás, gertom@inf.u-szeged.hu
 * 2014. Október 15. Gergely Tamás, gertom@inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

#define L 10                                        /* lapméret */

typedef struct {                                  /* a tömb elemtípusa */
    char nev[32];
    float adat;
} elem_t;

typedef elem_t *lap_t;

typedef lap_t  *lapterkep_t;

typedef unsigned int index_t;
```

Adatok rendezése

beszuro.c (v1.0) [29–50]

```
29 typedef struct flex_tomb_t {
30     lapterkep_t lt;                                    /* laptérkép */
31     index_t     hatar;                          /* aktuális indexhatár */
32 } flex_tomb_t;
33
34 /* A műveletek megvalósítása: */
35
36 void kiolvas(flex_tomb_t a, index_t i, elem_t *x) {
37     if (i < a.hatar) {
38         *x = a.lt[i / L][i % L];
39     }
40 }
41
42 void modosit(flex_tomb_t a, index_t i, elem_t x) {
43     if (i < a.hatar) {
44         a.lt[i / L][i % L] = x;
45     }
46 }
47
48 index_t felso(flex_tomb_t a) {
49     return a.hatar;
50 }
```

Adatok rendezése

beszuro.c (v1.0) [52–75]

```
52 void letesit(flex_tomb_t *a, unsigned int n) {
53     a->hatar = n;
54     if (n) {
55         int j;
56         a->lt = (elem_t**) malloc((1 + ((n - 1) / L)) * sizeof(lap_t));
57         for (j = 0; j <= ((n - 1) / L); ++j) {                /* lapok létesítése */
58             a->lt[j] = (elem_t*) malloc(L * sizeof(elem_t));
59         }
60     } else {
61         a->lt = NULL;
62     }
63 }
64
65 void megszuntet(flex_tomb_t *a) {
66     if (a->hatar) {
67         int j;
68         for (j = 0; j <= ((a->hatar - 1) / L); ++j) {            /* lapok törlése */
69             free(a->lt[j]);
70         }
71         free(a->lt);
72         a->lt = NULL;
73         a->hatar = 0;
74     }
75 }
```

Adatok rendezése

beszuro.c (v1.0) [77–98]

```
77 void novel(flex_tomb_t *a, index_t d) {
78     int j;
79     a->lt = (elem_t**) realloc(a->lt,
80                                (1 + ((a->hatar + d - 1) / L)) * sizeof(lap_t));
81     for (j = ((a->hatar) ? ((a->hatar - 1) / L) + 1 : 0);           /*   új lapok */
82                                j <= (a->hatar + d - 1) / L; ++j) {     /* létesítése */
83         a->lt[j] = (elem_t*) malloc(L * sizeof(elem_t));
84     }
85     a->hatar += d;
86 }
87
88 void csokkent(flex_tomb_t *a, index_t d) {
89     if (d <= a->hatar) {
90         int j;
91         for (j = (a->hatar - d - 1) / L + 1; j <= (a->hatar - 1) / L; ++j) {
92             free(a->lt[j]);                              /* felesleges lapok törlése */
93         }
94         a->hatar -= d;
95         a->lt = (elem_t**) realloc(a->lt,
96                                (1 + ((a->hatar - 1) / L)) * sizeof(lap_t));
97     }
98 }
```

Adatok rendezése

beszuro.c (v1.0) [100–124]

```
100 /* Rendezés */
101
102 typedef bool (*rend_rel_t)(elem_t, elem_t);
103
104 void beszuro_rendezes(flex_tomb_t t, rend_rel_t kisebb) {
105                           /* A kisebb rendezési reláció szerinti helyben rendezés */
106     int    i, j;
107     elem_t e, f;
108     for (i = 1; i < felso(t); ++i) {
109         kiolvas(t, i, &e);
110         j = i - 1;
111
112         while (true) {
113             if (j < 0) {
114                 break;
115             }
116             kiolvas(t, j, &f);
117             if (kisebb(f, e)) {
118                 break;
119             }
120             modosit(t, ((j--) + 1), f);
121         }
122         modosit(t, j + 1, e);
123     }
124 } /* beszuro_rendezes */
```

Adatok rendezése

beszuro.c (v1.0) [126–139]

```
126 bool rend_nev_novekvo(elem_t x, elem_t y) {
127                                          /* a névsor szerinti rendezési reláció  */
128     return strcmp(x.nev, y.nev) <= 0;
129 }
130
131 bool rend_adat_novekvo(elem_t x, elem_t y) {
132                                           /* az adat szerinti rendezési reláció  */
133     return x.adat <= y.adat;
134 }
135
136 bool rend_adat_csokkeno(elem_t x, elem_t y) {
137                                        /* az adat szerint csökkenő rendezési reláció  */
138     return x.adat >= y.adat;
139 }
```

Adatok rendezése

beszuro.c (v1.0) [141–162]

```
141 void read_sor(flex_tomb_t *t) {
142     index_t i = 0;
143     elem_t hallg;
144     printf("Kérem␣az␣adatsort,␣külön␣sorban␣név␣és␣adat!\n");
145     printf("A␣végét␣a␣*␣jelzi.\n");
146     scanf("%20[^\n]%*[^\n]", hallg.nev); getchar();
147     while (strcmp(hallg.nev, "*")) {
148         novel(t, 1);
149         scanf("%f%*[^\n]", &hallg.adat); getchar();
150         modosit(*t, i++, hallg);
151         scanf("%20[^\n]%*[^\n]", hallg.nev); getchar();
152     }
153 }
154
155 void write_sor(flex_tomb_t t) {
156     elem_t e;
157     index_t i;
158     for (i = 0; i < felso(t); ++i) {
159         kiolvas(t, i, &e);
160         printf("%6.2f␣%s\n", e.adat, e.nev);
161     }
162 }
```

Adatok rendezése

beszuro.c (v1.0) [164–184]

```
164 int main() {
165     flex_tomb_t sor;
166 
167     letesit(&sor, 0);                                   /* a flexibilis tömb létesítése */
168     read_sor(&sor);
169 
170     beszuro_rendezes(sor, rend_nev_novekvo);                  /* Rend. névsor szerint */
171     printf("Névsor␣szerint␣rendezve:\n");
172     write_sor(sor);                                                    /* Kiíratás */
173 
174     beszuro_rendezes(sor, rend_adat_novekvo);                   /* Rend. adat szerint */
175     printf("Adat␣szerint␣rendezve:\n");
176     write_sor(sor);                                                    /* Kiíratás */
177 
178     beszuro_rendezes(sor, rend_adat_csokkeno);                      /* Rendezés újra */
179     printf("Adat␣szerint␣csökkenő␣sorba␣rendezve:\n");
180     write_sor(sor);                                                    /* Kiíratás */
181 
182     megszuntet(&sor);                                 /* a flexibilis tömb törlése */
183     return 0;
184 }
```

Előjeles/előjeltelen karaktertípus ellenőrzése [1/1]

char.c [1–14]

```c
/* A char pontos típusának ellenőrzése.
 * 2006. Augusztus 8.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    char a;
    unsigned char b;
    a = b = 128;
    a >>= 1; b >>= 1;
    printf("Ezen a gépen %ssigned char van\n", ((a == b) ? "un" : ""));
    return 0;
}
```

Konstans változó értékének megváltoztatása [1/1]

const.c [1–14]

```c
/* Egy const értékének megváltoztatása.
 * 2006. Augusztus 17. Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    const int c = 100;
    int *p;
    p = (int*)&c;
    *p *= 2;
    printf("%d\n", c);
    return 0;
}
```

Csúszóátlag számítása [1/1]

csuszoatlag.c [1–25]

```c
/* Csúszóátlag számítása fix méretű tömbön.
 * 2012. Szeptember 5.  Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main() {
    double ertektomb[N], atlagtomb[N-2];
    int i;
    for (i = 0; i < N; ++i) {
        printf("? "); scanf("%lf", &(ertektomb[i]));
    }
    for (i = 0; i < N-2; ++i)
        atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
    for (i = 0; i < N-2; ++i)
        printf("%lf;", atlagtomb[i]);
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítása adott elemszámra

csuszoatlag.c (v1.0) [1–26]

```c
/* Csúszóátlag számítása fix méretű tömbön.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define N 15

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main() {
    double ertektomb[N], atlagtomb[N - 2];
    for (int i = 0; i < N; ++i) {
        printf("? "); scanf("%lf", &(ertektomb[i]));
    }
    for (int i = 0; i < N - 2; ++i)
        atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
    for (int i = 0; i < N - 2; ++i)
        printf("%lf;", atlagtomb[i]);
    putchar('\n');
    return 0;
}
```

Csúszóátlag számítás parancssorból

csuszoatlag.c (v2.0) [1–29]

```c
/* Csúszóátlag számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

double atlag3(double v1, double v2, double v3) {
    return (v1 + v2 + v3) / 3;
}

int main(int argc, char *argv[]) {
    double ertektomb[3];
    if (argc < 4) {
        return 1;
    }
    ertektomb[0] = atof(argv[1]);
    ertektomb[1] = atof(argv[2]);
    for (int i = 3; i < argc; ++i) {
        ertektomb[2] = atof(argv[i]);
        printf("%lf;", atlag3(ertektomb[0], ertektomb[1], ertektomb[2]));
        ertektomb[0] = ertektomb[1];
        ertektomb[1] = ertektomb[2];
    }
    putchar('\n');
    return 0;
}
```

Fájlkezelés text és binary módban [1/1]

data.c [1–27]

```c
/* Fájlkezelés többféle módon.
 *  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    FILE *fp, *fpb;
    int a, b;

    fp = fopen("probat.txt", "wt");
    fprintf(fp, "%d", 34);
    fflush(fp); fclose(fp);

    fp = fopen("probat.txt", "rt");
    fscanf(fp, "%d", &a);
    printf("%d\n", a);
    fclose(fp);

    fpb = fopen("probab.dat", "w+b");
    fwrite(&a, sizeof(int), 1, fpb);
    fseek(fpb, 0, SEEK_SET);
    fread(&b, sizeof(int), 1, fpb);
    printf("%d\n", b);
    fclose(fpb);
    return 0;
}
```

Dátum helyességének eldöntése [1/3]

datum.c [1–12]

```c
/* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
 * 1997. Október 4.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    int honap, nap;
    int jo;                                  /* a Boolean érték tárolására */

    printf("Kérem a dátumot (hónap, nap) !\n");
    scanf("%d%d", &honap, &nap);
```

Dátum helyességének eldöntése [2/3]

datum.c [14–36]

```
14      switch (honap) {
15      case 2:
16          jo = (1 <= nap && nap <= 28);
17          break;
18      case 4:
19      case 6:
20      case 9:
21      case 11:
22          jo = (1 <= nap && nap <= 30);
23          break;
24      case 1:
25      case 3:
26      case 5:
27      case 7:
28      case 8:
29      case 10:
30      case 12:
31          jo = (1 <= nap && nap <= 31);
32          break;
33      default:
34          jo = 0;
35          break;
36      } /* switch */
```

Dátum helyességének eldöntése [3/3]

datum.c [37–44]

```
37                                                          /* Kiíratás  */
38      printf("A␣dátum␣");
39      if (!jo) {                                      /* Ezt másképpen szokták */
40          printf("nem␣");
41      }
42      printf("helyes.\n");
43      return 0;
44  }
```

Helyes-e a dátum (nem szökőévben)?

datum.c (v1.0) [1–7]

```c
/* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
 *    1997. Október 4.  Dévényi Károly, devenyi@inf.u-szeged.hu
 *    2018. Június 17.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

Helyes-e a dátum (nem szökőévben)?

datum.c (v1.0) [9–35]

```
 9 bool datum(int honap, int nap) {
10     int utolso;
11     switch (honap) {
12     case 2:
13         utolso = 28;
14         break;
15     case 4:
16     case 6:
17     case 9:
18     case 11:
19         utolso = 30;
20         break;
21     case 1:
22     case 3:
23     case 5:
24     case 7:
25     case 8:
26     case 10:
27     case 12:
28         utolso = 31;
29         break;
30     default:
31         utolso = 0;
32         break;
33     } /* switch */
34     return 1 <= nap && nap <= utolso;
35 }
```

Helyes-e a dátum (nem szökőévben)?

datum.c (v1.0) [37–47]

```
37 void kiir(int honap, int nap) {
38     printf(datum(honap, nap) ? "A␣dátum␣helyes.\n" : "A␣dátum␣nem␣helyes.\n");
39 }
40
41 int main() {
42     int honap, nap;
43     printf("Kérem␣a␣dátumot␣(hónap,␣nap)␣!\n");
44     scanf("%d%d", &honap, &nap);
45     kiir(honap, nap);
46     return 0;
47 }
```

Helyes-e a dátum (nem szökőévben)?

datum.c (v2.0) [1–12]

```c
/* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
 *    1997. Október 4.  Dévényi Károly, devenyi@inf.u-szeged.hu
 *    2018. Június 17.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdbool.h>
#include <stdio.h>

typedef struct {
    int honap;
    int nap;
} datum_t;
```

Helyes-e a dátum (nem szökőévben)?

datum.c (v2.0) [14–40]

```
14 bool datum(datum_t d) {
15     int utolso;
16     switch (d.honap) {
17     case 2:
18         utolso = 28;
19         break;
20     case 4:
21     case 6:
22     case 9:
23     case 11:
24         utolso = 30;
25         break;
26     case 1:
27     case 3:
28     case 5:
29     case 7:
30     case 8:
31     case 10:
32     case 12:
33         utolso = 31;
34         break;
35     default:
36         utolso = 0;
37         break;
38     } /* switch */
39     return 1 <= d.nap && d.nap <= utolso;
40 }
```

Helyes-e a dátum (nem szökőévben)?

datum.c (v2.0) [42–56]

```
42 void beolvas(datum_t *d) {
43     scanf("%d%d", &d->honap, &d->nap);
44 }
45
46 void kiir(datum_t d) {
47     printf(datum(d) ? "A dátum helyes.\n" : "A dátum nem helyes.\n");
48 }
49
50 int main() {
51     datum_t d;
52     printf("Kérem a dátumot (hónap, nap)!\n");
53     beolvas(&d);
54     kiir(d);
55     return 0;
56 }
```

Helyes-e a dátum (évvel együtt)?

datum.c (v3.0) [1–13]

```c
/* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
 *    1997. Október 4.  Dévényi Károly, devenyi@inf.u-szeged.hu
 *    2018. Június 17.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdbool.h>
#include <stdio.h>

typedef struct {
    int ev;
    int honap;
    int nap;
} datum_t;
```

Helyes-e a dátum (évvel együtt)?

datum.c (v3.0) [15–41]

```
15 bool datum(datum_t d) {
16     int utolso;
17     switch (d.honap) {
18     case 2:
19         utolso = ((d.ev % 4) ? 28 : (d.ev % 100) ? 29 : (d.ev % 400) ? 28 : 29);
20         break;
21     case 4:
22     case 6:
23     case 9:
24     case 11:
25         utolso = 30;
26         break;
27     case 1:
28     case 3:
29     case 5:
30     case 7:
31     case 8:
32     case 10:
33     case 12:
34         utolso = 31;
35         break;
36     default:
37         utolso = 0;
38         break;
39     } /* switch */
40     return 1 <= d.nap && d.nap <= utolso;
41 }
```

Helyes-e a dátum (évvel együtt)?

datum.c (v3.0) [43–57]

```
43 void beolvas(datum_t *d) {
44     scanf("%d.%d.%d.", &d->ev, &d->honap, &d->nap);
45 }
46
47 void kiir(datum_t d) {
48     printf(datum(d) ? "A dátum helyes.\n" : "A dátum nem helyes.\n");
49 }
50
51 int main() {
52     datum_t d;
53     printf("Kérem a dátumot (év.hónap.nap.)!\n");
54     beolvas(&d);
55     kiir(d);
56     return 0;
57 }
```

Helyesek-e a parancssorban felsorolt dátumok?

datum.c (v4.0) [1–13]

```c
/* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
 *    1997. Október 4.  Dévényi Károly, devenyi@inf.u-szeged.hu
 *    2018. Június 17.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdbool.h>
#include <stdio.h>

typedef struct {
    int ev;
    int honap;
    int nap;
} datum_t;
```

Helyesek-e a parancssorban felsorolt dátumok?

datum.c (v4.0) [15–41]

```
15 bool datum(datum_t d) {
16     int utolso;
17     switch (d.honap) {
18     case 2:
19         utolso = ((d.ev % 4) ? 28 : (d.ev % 100) ? 29 : (d.ev % 400) ? 28 : 29);
20         break;
21     case 4:
22     case 6:
23     case 9:
24     case 11:
25         utolso = 30;
26         break;
27     case 1:
28     case 3:
29     case 5:
30     case 7:
31     case 8:
32     case 10:
33     case 12:
34         utolso = 31;
35         break;
36     default:
37         utolso = 0;
38         break;
39     } /* switch */
40     return 1 <= d.nap && d.nap <= utolso;
41 }
```

Helyesek-e a parancssorban felsorolt dátumok?

datum.c (v4.0) [43–70]

```
43 void str_to_datum(const char *str, datum_t *d) {
44     sscanf(str, "%d.%d.%d.", &d->ev, &d->honap, &d->nap);
45 }
46
47 void beolvas(datum_t *d) {
48     scanf("%d.%d.%d.", &d->ev, &d->honap, &d->nap);
49 }
50
51 void kiir(datum_t d) {
52     printf("A␣%d.%02d.%02d.␣dátum%shelyes.\n",
53             d.ev, d.honap, d.nap, datum(d) ? "␣" : "␣nem␣");
54 }
55
56 int main(int argc, char *argv[]) {
57     datum_t d;
58     if (argc < 2) {
59         printf("Kérem␣a␣dátumot␣(év.hónap.nap.)!\n");
60         beolvas(&d);
61         kiir(d);
62     } else {
63         int i;
64         for (i = 1; i < argc; ++i) {
65             str_to_datum(argv[i], &d);
66             kiir(d);
67         }
68     }
69     return 0;
70 }
```

Forgó fánk [1/1]

donut.c [1–21]

```
1                k;double sin()
2             ,cos();main(){float A=
3           0,B=0,i,j,z[1760];char b[
4         1760];printf("\x1b[2J");for(;;
5      ){memset(b,32,1760);memset(z,0,7040)
6      ;for(j=0;6.28>j;j+=0.07)for(i=0;6.28
7     >i;i+=0.02){float c=sin(i),d=cos(j),e=
8     sin(A),f=sin(j),g=cos(A),h=d+2,D=1/(c*
9     h*e+f*g+5),l=cos      (i),m=cos(B),n=s\
10    in(B),t=c*h*g-f*        e;int x=40+30*D*
11    (l*h*m-t*n),y=            12+15*D*(l*h*n
12    +t*m),o=x+80*y,          N=8*((f*e-c*d*g
13     )*m-c*d*e-f*g-l        *d*n);if(22>y&&
14     y>0&&x>0&&80>x&&D>z[o]){z[o]=D;;;b[o]=
15     ".,-~:;=!*#$@"[N>0?N:0];}}/*#****!!-*/
16      printf("\x1b[H");for(k=0;1761>k;k++)
17       putchar(k%80?b[k]:10);A+=0.04;B+=
18         0.02;}}/*****##########!!=;:~
19           ~::==!!!**********!!!==::-
20             .,~~;;;========;;;:~-.
21                 ..,--------,*/
```

Eltelt idő kiszámítása [1/1]

eltelt.c [1–24]

```c
/* Egy nap két időpontja között mennyi idő telt el.
 * 1997. Szeptember 26.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    int o1, p1;                          /* az első időpont */
    int o2, p2;                         /* a második időpont */
    int o, p;                              /* az eltelt idő */
    int k;                           /* az eltelt idő percben */
    /* beolvasás */
    printf("Kérem az első időpontot óra perc formában\n");
    scanf("%d %d", &o1, &p1);
    printf("Kérem a második időpontot óra perc formában\n");
    scanf("%d %d", &o2, &p2);
    /*  számítás */
    k = 60 * o2 + p2 - (60 * o1 + p1);
    o = k / 60;
    p = k % 60;
    /*  kiíratás */
    printf("Az eltelt idő: %d óra %d perc.\n", o, p);
    return 0;
}
```

Eltelt idő kiszámítása [1/2]

eltelt-arg.c [1–25]

```c
/* Egy nap két időpontja között mennyi idő telt el.
 * 2013. November 7.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

typedef struct ido_t {
    int ora;
    int perc;
} ido_t;

ido_t str_to_ido(const char * str) {
    ido_t ret = {0, 0};
    ret.ora  = atoi(str);
    for (; *str && *str != ':'; ++str);
    if (*str) {
        ret.perc = atoi(str + 1);
    }
    return ret;
}

int ido_to_int(ido_t t) {
    return 60 * t.ora + t.perc;
}
```

Eltelt idő kiszámítása [2/2]

eltelt-arg.c [27–48]

```
27 ido_t int_to_ido(int t) {
28     ido_t ret;
29     ret.ora  = t / 60;
30     ret.perc = t % 60;
31     return ret;
32 }
33
34 ido_t eltelt_ido(ido_t t1, ido_t t2) {
35     return int_to_ido(ido_to_int(t2) - ido_to_int(t1));
36 }
37
38 int main(int argc, char *argv[]) {
39     ido_t t1, t2, dt;
40     if (argc < 3) {
41         return 1;
42     }
43     t1 = str_to_ido(argv[1]);
44     t2 = str_to_ido(argv[2]);
45     dt = eltelt_ido(t1, t2);
46     printf("Az␣eltelt␣idő␣%d:%02d\n", dt.ora, dt.perc);
47     return 0;
48 }
```

Eltelt idő kiszámítása [1/1]

eltelt-fgv.c [1–26]

```c
/* Egy nap két időpontja között mennyi idő telt el.
 * 2007. Augusztus 30.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

int eltelt_percek(int ora1, int perc1, int ora2, int perc2) {
    return 60 * ora2 + perc2 - (60 * ora1 + perc1);
}

int main() {
    int o1, p1;                         /* az első időpont */
    int o2, p2;                        /* a második időpont */
    int o, p;                             /* az eltelt idő */
    /* beolvasás */
    printf("Kérem az első időpontot óra perc formában\n");
    scanf("%d %d", &o1, &p1);
    printf("Kérem a második időpontot óra perc formában\n");
    scanf("%d %d", &o2, &p2);
    /* számítás */
    o = eltelt_percek(o1, p1, o2, p2) / 60;
    p = eltelt_percek(o1, p1, o2, p2) % 60;
    /* kiíratás */
    printf("Az eltelt idő: %d óra %d perc.\n", o, p);
    return 0;
}
```

Eltelt idő kiszámítása [1/2]

eltelt-struct.c [1–25]

```c
/* Egy nap két időpontja között mennyi idő telt el.
 * 2013. November 7.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct ido_t {
    int ora;
    int perc;
} ido_t;

int ido_to_int(ido_t t) {
    return 60 * t.ora + t.perc;
}

ido_t int_to_ido(int t) {
    ido_t ret;
    ret.ora  = t / 60;
    ret.perc = t % 60;
    return ret;
}

ido_t eltelt_ido(ido_t t1, ido_t t2) {
    return int_to_ido(ido_to_int(t2) - ido_to_int(t1));
}
```

Eltelt idő kiszámítása [2/2]

eltelt-struct.c [27–39]

```c
int main() {
    ido_t t1, t2, dt;
    /* beolvasás */
    printf("Kérem az első időpontot óra perc formában\n");
    scanf("%d %d", &t1.ora, &t1.perc);
    printf("Kérem a második időpontot óra perc formában\n");
    scanf("%d %d", &t2.ora, &t2.perc);
    /* számítás */
    dt = eltelt_ido(t1, t2);
    /* kiíratás */
    printf("Az eltelt idő: %d óra %d perc.\n", dt.ora, dt.perc);
    return 0;
}
```

Nap két időpontja között eltelt idő

eltelt.c (v1.0) [1–19]

```c
/* Egy nap két időpontja között mennyi idő telt el.
 *
 * 1997. Szeptember 26. Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

void eltelt(int o1, int p1, int o2, int p2) {
    int o, p;                              /* az eltelt idő */
    o = o2 - o1;
    p = p2 - p1;
    if (p < 0) {
        o -= 1;
        p += 60;
    }
    printf("%d:%02d␣és␣%d:%02d␣között␣eltelt␣idő␣%d:%02d\n",
           o1,  p1,   o2,  p2,                  o,   p);
}
```

Nap két időpontja között eltelt idő

eltelt.c (v1.0) [21–31]

```
21 int main() {
22     int o1, p1;                          /* az első időpont */
23     int o2, p2;                         /* a második időpont */
24     /* beolvasás */
25     printf("Kérem az első időpontot óra perc formában\n");
26     scanf("%d %d", &o1, &p1);
27     printf("Kérem a második időpontot óra perc formában\n");
28     scanf("%d %d", &o2, &p2);
29     eltelt(o1, p1, o2, p2);
30     return 0;
31 }
```

Nap két időpontja között eltelt idő

eltelt.c (v2.0) [1–26]

```c
/* Egy nap két időpontja között mennyi idő telt el.
 *
 * 1997. Szeptember 26. Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int eltelt(int o1, int p1, int o2, int p2) {
    return 60 * o2 + p2 - (60 * o1 + p1);
}

int main() {
    int o1, p1;                         /* az első időpont */
    int o2, p2;                        /* a második időpont */
    int k;                                /* az eltelt idő */
    /* beolvasás */
    printf("Kérem az első időpontot óra perc formában\n");
    scanf("%d %d", &o1, &p1);
    printf("Kérem a második időpontot óra perc formában\n");
    scanf("%d %d", &o2, &p2);
    k = eltelt(o1, p1, o2, p2);
    printf("%d:%02d és %d:%02d között eltelt idő %d:%02d\n",
            o1,  p1,   o2,  p2,              k / 60, k % 60);
    return 0;
}
```

Nap két időpontja között eltelt idő

eltelt.c (v3.0) [1–27]

```c
/* Egy nap két időpontja között mennyi idő telt el.
 *
 * 2013. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct ido_t {
    int ora;
    int perc;
} ido_t;

int ido_to_int(ido_t t) {
    return 60 * t.ora + t.perc;
}

ido_t int_to_ido(int t) {
    ido_t ret = {0, 0};
    ret.ora  = t / 60;
    ret.perc = t % 60;
    return ret;
}

ido_t eltelt_ido(ido_t t1, ido_t t2) {
    return int_to_ido(ido_to_int(t2) - ido_to_int(t1));
}
```

Nap két időpontja között eltelt idő

eltelt.c (v3.0) [29–39]

```
29 int main() {
30     ido_t t1, t2, dt;
31     printf("Kérem␣az␣első␣időpontot␣óra␣perc␣formában\n");
32     scanf("%d␣%d", &t1.ora, &t1.perc);
33     printf("Kérem␣a␣második␣időpontot␣óra␣perc␣formában\n");
34     scanf("%d␣%d", &t2.ora, &t2.perc);
35     dt = eltelt_ido(t1, t2);
36     printf("%d:%02d␣és␣%d:%02d␣között␣eltelt␣idő␣%d:%02d\n",
37         t1.ora, t1.perc, t2.ora, t2.perc, dt.ora, dt.perc);
38     return 0;
39 }
```

Nap két időpontja között eltelt idő (óra:perc formátum)

eltelt.c (v4.0) [1–27]

```c
/* Egy nap két időpontja között mennyi idő telt el.
 *
 * 2013. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

typedef struct ido {
    int ora;
    int perc;
} ido;

ido str_to_ido(const char * str) {
    ido ret = {0, 0};
    ret.ora  = atoi(str);
    for (; *str && *str != ':'; ++str);
    if (*str) {
        ret.perc = atoi(++str);
    }
    return ret;
}

int ido_to_int(ido t) {
    return 60 * t.ora + t.perc;
}
```

Nap két időpontja között eltelt idő (óra:perc formátum)

eltelt.c (v4.0) [29–51]

```
29 ido int_to_ido(int t) {
30     ido ret = {0, 0};
31     ret.ora  = t / 60;
32     ret.perc = t % 60;
33     return ret;
34 }
35
36 ido eltelt_ido(ido t1, ido t2) {
37     return int_to_ido(ido_to_int(t2) - ido_to_int(t1));
38 }
39
40 int main(int argc, char *argv[]) {
41     ido t1, t2, dt;
42     if (argc < 3) {
43         return 1;
44     }
45     t1 = str_to_ido(argv[1]);
46     t2 = str_to_ido(argv[2]);
47     dt = eltelt_ido(t1, t2);
48     printf("%d:%02d␣és␣%d:%02d␣között␣eltelt␣idő␣%d:%02d\n",
49         t1.ora, t1.perc, t2.ora, t2.perc, dt.ora, dt.perc);
50     return 0;
51 }
```

Nap két időpontja között eltelt idő (parancssorból)

eltelt.c (v5.0) [1–28]

```c
/* Egy nap két időpontja között mennyi idő telt el.
 *
 * 2013. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

typedef struct ido {
    int ora;
    int perc;
    int masodperc;
} ido;

ido str_to_ido(const char * str) {
    ido ret = {0, 0, 0};
    ret.ora  = atoi(str);
    for (; *str && *str != ':'; ++str);
    if (*str) {
        ret.perc = atoi(++str);
    }
    for (; *str && *str != ':'; ++str);
    if (*str) {
        ret.masodperc = atoi(++str);
    }
    return ret;
}
```

Nap két időpontja között eltelt idő (parancssorból)

eltelt.c (v5.0) [30–58]

```
30 int ido_to_int(ido t) {
31     return 3600 * t.ora + 60 * t.perc + t.masodperc;
32 }
33
34 ido int_to_ido(int t) {
35     ido ret = {0, 0, 0};
36     ret.ora       = t / 3600;
37     ret.perc      = t / 60 % 60;
38     ret.masodperc = t % 60;
39     return ret;
40 }
41
42 ido eltelt_ido(ido t1, ido t2) {
43     return int_to_ido(ido_to_int(t2) - ido_to_int(t1));
44 }
45
46 int main(int argc, char *argv[]) {
47     ido t1, t2, dt;
48     if (argc < 3) {
49         return 1;
50     }
51     t1 = str_to_ido(argv[1]);
52     t2 = str_to_ido(argv[2]);
53     dt = eltelt_ido(t1, t2);
54     printf("%d:%02d:%02d␣és␣%d:%02d:%02d␣között␣eltelt␣idő␣%d:%02d:%02d\n",
55         t1.ora, t1.perc, t1.masodperc, t2.ora, t2.perc, t2.masodperc,
56         dt.ora, dt.perc, dt.masodperc);
57     return 0;
58 }
```

Architektúra bájtsorrendjének meghatározása [1/1]

endian.c [1–16]

```c
/* Eldönti, hogy milyen a bájtsorrend a számítógépen.
 * 1998. Április 24.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    int x = 1;

    if (*(char *)&x == 1) {
        printf("little-endian\n");
    } else {
        printf("big-endian\n");
    }
    return 0;
}
```

A nappalis gyakorlat értékelése [1/9]

eredmeny.c [1–25]

```c
/* Programozás alapjai gyakorlat értékelés.
 *
 * A program a parancssorban sorban megadott zh és plusz pontok alapján
 * kiszámolja a várható érdemjegyet.
 *
 * 2021. Január 27.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

#ifndef SILENT
#define GR "A gyakorlat eredménye: "
#else
#define GR
#endif
#define G1 "elégtelen(1)"
#define G2 "elégséges(2)"
#define G3 "közepes(3)"
#define G4 "jó(4)"
#define G5 "jeles(5)"

#define MAX_ZH_PER_WEEK 2
```

A nappalis gyakorlat értékelése [2/9]

eredmeny.c [27–54]

```
27 #define OPT_HELP_S "-h"
28 #define OPT_HELP_L "--help"
29 #define OPT_NO_MIN "--no-minimums"
30
31 typedef struct {
32     int   min;
33     char *jegy;
34 } eredmeny_t;
35
36 const eredmeny_t G[] = {
37     {80, G5},
38     {70, G4},
39     {57, G3},
40     {45, G2},
41     { 0, G1},
42     { 0, NULL}
43 };
44
45 const int ZHPWEEK[] = {  1,  1,  1,  1,  1,  1,  1,  1,  2,  0 };
46 const int MINIMUM[] = {  1,  4,  7, 10, 15, 20, 25, 30, 45 };
47 const int ZHMAXPT[] = {  5,  5,  5,  5, 10, 10, 10, 10, 10, 10 };
48 const int MAXPLUS   = 5;
49 const int MAXPRGP   = 5;
50 const int NINCSMIN  = 1;
51 const int NINCSVED  = 1;
52
53 bool minimum_ellenorzes = true;
54 int  vedes_nem_sikerult = 0;
```

A nappalis gyakorlat értékelése [3/9]

eredmeny.c [56–78]

```
56 int heti_pont(int w, int zhn, int zhp[], int hfp[]) {
57     int sum = 0;
58     int db = ZHPWEEK[w];
59     for (int i = 0, j = 0; i < db; ++i, ++zhn) {
60         if (zhp[i] < 0) {
61 #ifndef SILENT
62             printf("Figyelmeztetés! A(z) %d. zh védése nincs elfogadva.\n", w + 1);
63 #endif
64             ++vedes_nem_sikerult;
65             zhp[i] = 0;
66         }
67         if (hfp[i] < 0) {
68 #ifndef SILENT
69             printf("Figyelmeztetés! A(z) %d. házi feladat védése nincs elfogadva.\n",
70                    w + 1);
71 #endif
72             ++vedes_nem_sikerult;
73             hfp[i] = 0;
74         }
75         sum += ((zhp[i] >= ZHMAXPT[zhn]) ? ZHMAXPT[zhn] : zhp[i]) + (!!hfp[j++]);
76     }
77     return sum;
78 }
```

A nappalis gyakorlat értékelése [4/9]

eredmeny.c [80–98]

```
80 bool min_bukas(int w, int zhp) {
81     static int nincs_min = 0;
82     if (minimum_ellenorzes && zhp < MINIMUM[w]) {
83 #ifndef SILENT
84         printf("Figyelmeztetés! A(z) %d. zh után nincs meg a minimális pontszám.\n"
85                "    elvárt: %d, meglévő: %d.\n", w + 1, MINIMUM[w], zhp);
86 #endif
87         if (++nincs_min > NINCSMIN) {
88 #ifndef SILENT
89             printf("A minimum pontszám kritériumait legalább %d alkalommal"
90                    " nem teljesítette, ami több, mint az elfogadható %d alkalom.\n",
91                    nincs_min, NINCSMIN);
92 #endif
93             printf(GR G1 "\n");
94             return true;
95         }
96     }
97     return false;
98 }
```

A nappalis gyakorlat értékelése [5/9]

eredmeny.c [100–123]

```
100 void usage(const char* progname) {
101     int db;
102     int maxpont = 0;
103     printf("Használat: %s [" OPT_HELP_S "|" OPT_HELP_L "|" OPT_NO_MIN "]", progname);
104     for (int w = 0; (db = ZHPWEEK[w]); ++w) {
105         if (db == 1) {
106             printf(" hf%d zh%d", w + 1, w + 1);
107         } else {
108             for (int i = 0; i < db; ++i) {
109                 printf(" hf%d/%d zh%d/%d", w + 1, i + 1, w + 1, i + 1);
110             }
111         }
112     }
113     printf(" plusz pprog\n"
114            "\n"
115            "    hfN: az N. heti házi feladat pontjai\n"
116            "    hfN/M: az N. heti zh M. házi feladat pontjai\n"
117            "    zhN: az N. heti zh pontszáma\n"
118            "    zhN/M: az N. heti zh M. részpontszáma\n"
119            "    plusz: plusz pontszám\n"
120            "    pprog: pluszprogram pontszám\n"
121            "\n"
122            "Összpontszámok minimuma (a plusz és pprog nem számítanak bele):\n"
123            "\n");
```

A nappalis gyakorlat értékelése [6/9]

eredmeny.c [124–148]

```
124     for (int w = 0, tmp = 0; (db = ZHPWEEK[w]); ++w) {
125         maxpont += db;
126         for (int i = 0; i < db; ++i) {
127             maxpont += ZHMAXPT[tmp++];
128         }
129         printf("␣␣␣␣%d.␣hét␣után␣%d␣pont␣(az␣addigi␣összesen␣%d␣pontból)\n",
130                w + 1, MINIMUM[w], maxpont);
131     }
132     maxpont += MAXPLUS + MAXPRGP;
133     printf("\n"
134            "Ha␣a␣minimum␣pontszámot␣több,␣mint␣%d␣alkalommal"
135            "␣nem␣sikerül␣teljesíteni,\n"
136            "vagy␣a␣leadott␣feladatok␣közül␣több,␣mint␣%d␣darabot"
137            "␣nem␣sikerül␣megvédeni,\n"
138            "akkor␣a␣félév␣végeredménye␣minden␣mástól␣függetlenül␣" G1 ".\n"
139            "\n"
140            "Értékelés,␣az␣összes␣részpont,␣plusz␣(maximum␣%d)␣és␣pprog␣(maximum␣%d)"
141            "␣összege␣alapján:\n"
142            "\n",
143            NINCSMIN, NINCSVED, MAXPLUS, MAXPRGP);
144     for (const eredmeny_t *p = G; p->jegy; ++p) {
145         printf("␣␣␣␣%3d␣--␣%3d␣pont:␣%s\n", p->min, maxpont, p->jegy);
146         maxpont = p->min - 1;
147     }
148 }
```

A nappalis gyakorlat értékelése [7/9]

eredmeny.c [150–173]

```
150 int main(int argc, char *argv[]) {
151     int sum = 0;
152     int plus = 0;
153     int prgp = 0;
154     int idx = 1;
155     int db;
156     int w, zhn;
157 
158     if (argc < 2) {
159 #ifndef SILENT
160         usage(argv[0]);
161 #endif
162         return 1;
163     }
164 
165     if (!strcmp(OPT_HELP_S, argv[1]) || !strcmp(OPT_HELP_L, argv[1])) {
166         usage(argv[0]);
167         return 0;
168     }
169 
170     if (!strcmp(OPT_NO_MIN, argv[1])) {
171         minimum_ellenorzes = false;
172         idx = 2;
173     }
```

A nappalis gyakorlat értékelése [8/9]

eredmeny.c [175–200]

```
175     for (w = 0, zhn = 0; (db = ZHPWEEK[w]) && (idx < argc); ++w, zhn += db) {
176         int hfp[MAX_ZH_PER_WEEK];
177         int zhp[MAX_ZH_PER_WEEK];
178         for (int i = 0; i < db; ++i) {
179             hfp[i] = (idx < argc) ? atoi(argv[idx]) : 0;
180             ++idx;
181             zhp[i] = (idx < argc) ? atoi(argv[idx]) : 0;
182             ++idx;
183         }
184         if (idx <= argc) {
185             if (min_bukas(w, sum += heti_pont(w, zhn, zhp, hfp))) {
186                 return 0;
187             }
188         } else {
189             break;
190         }
191     }
192
193     if (vedes_nem_sikerult > NINCSVED) {
194#ifndef SILENT
195         printf("A leadott feladata %d alkalommal nem lett elfogadva, ami több,"
196                " mint az elfogadható %d alkalom.\n", vedes_nem_sikerult, NINCSVED);
197#endif
198         printf(GR G1 "\n");
199         return 0;
200     }
```

A nappalis gyakorlat értékelése [9/9]

eredmeny.c [202–229]

```
202     if (db > 0) {  // Ha még nincs pontszám minden zh-hoz (break volt)
203#ifndef SILENT
204         printf("%d␣zh␣eredménye␣alapján␣jelenleg␣%d␣pontja␣van.\n", w, sum);
205#endif
206         return 0;
207     }
208
209     if (idx < argc) { // ha van plusz pont is
210         plus = atoi(argv[idx++]);
211     }
212     sum += (plus > MAXPLUS) ? MAXPLUS : plus;
213
214     if (idx < argc) { // ha van program plusz pont is
215         prgp = atoi(argv[idx++]);
216     }
217     sum += (prgp > MAXPRGP) ? MAXPRGP : prgp;
218
219#ifndef SILENT
220     printf("A␣végső␣pontszáma%s␣%d.\n", (idx < argc) ? "␣legalább" : "", sum);
221#endif
222     for (const eredmeny_t *p = G; p->jegy; ++p) { // A pontszámhoz tartozó jegy keresése
223         if (sum >= p->min) {
224             printf(GR "%s\n", p->jegy);
225             return 0;
226         }
227     }
228     return -1;
229 }
```

Egy dátum az év hányadik napja? [1/2]

evnapja.c [1–11]

```c
/* Egy YYYY-MM-DD formátumban megadott dátum az év hányadik napja?
 * 2012. Szeptember 5.  Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct {
    int ev;
    int ho;
    int nap;
} datum_t;
```

Egy dátum az év hányadik napja? [2/2]

evnapja.c [13–36]

```c
int main() {
    datum_t d;
    int ev_napja;

    scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
    ev_napja = d.nap;
    switch (d.ho) {
        case 12: ev_napja += 30;
        case 11: ev_napja += 31;
        case 10: ev_napja += 30;
        case  9: ev_napja += 31;
        case  8: ev_napja += 31;
        case  7: ev_napja += 30;
        case  6: ev_napja += 31;
        case  5: ev_napja += 30;
        case  4: ev_napja += 31;
        case  3: ev_napja += (d.ev%4)?28:((d.ev%100)?29:((d.ev%400)?28:29));
        case  2: ev_napja += 31;
        case  1: break;
        default: return 1;
    }
    printf("%d-%d-%d␣az␣év␣%d.␣napja\n", d.ev, d.ho, d.nap, ev_napja);
    return 0;
}
```

Egy dátum az év hányadik napja?

evnapja.c (v1.0) [1–13]

```c
/* Egy YYYY-MM-DD formátumban megadott dátum az év hányadik napja?
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

typedef struct {
    int ev;
    int ho;
    int nap;
} datum_t;
```

Egy dátum az év hányadik napja?

evnapja.c (v1.0) [15–43]

```
15 int main() {
16     datum_t d;
17     int ev_napja;
18
19     scanf("%d-%d-%d", &d.ev, &d.ho, &d.nap);
20     ev_napja = d.nap;
21     switch (d.ho) {
22         case 12: ev_napja += 30;
23         case 11: ev_napja += 31;
24         case 10: ev_napja += 30;
25         case  9: ev_napja += 31;
26         case  8: ev_napja += 31;
27         case  7: ev_napja += 30;
28         case  6: ev_napja += 31;
29         case  5: ev_napja += 30;
30         case  4: ev_napja += 31;
31         case  3: ev_napja += (d.ev % 4) ?
32                              28 : ((d.ev % 100) ?
33                                    29 : ((d.ev % 400) ?
34                                          28 : 29
35                                    )
36                              );
37         case  2: ev_napja += 31;
38         case  1: break;
39         default: return 1;
40     }
41     printf("%d-%d-%d az év %d. napja\n", d.ev, d.ho, d.nap, ev_napja);
42     return 0;
43 }
```

Függvény határozott integráljának kiszámítása [1/5]

fgvint.c [1–24]

```c
/* Közelítő integrálás a trapéz szabály segítségével.
 * Az integrálandó függvény nevét a parancssorból kapjuk.
 * Meg kell adnunk az intervallumot is és a finomságot is.
 * Kérhető help is.
 * 1998. Április 14.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2020. Július 24. Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <errno.h>

typedef struct tablaelem_t {
    char *nev;                          /* A függvény neve szövegesen  */
    double (*fuggveny)();               /* A függvényre mutató pointer */
} tablaelem_t;

static double help();

static double expx();

static double sin2x();
```

Függvény határozott integráljának kiszámítása [2/5]

fgvint.c [26–42]

```
26 tablaelem_t tablazat[] = {
27     { "sin",   sin   },
28     { "tan",   tan   },
29     { "cos",   cos   },
30     { "expx",  expx  },
31     { "sin2x", sin2x },
32     { "help",  help  },
33     {  NULL,   NULL  }                    /* A táblázat végét jelzi */
34 };
35
36 static double expx(double x) {
37     return (exp(x) / x);
38 }
39
40 static double sin2x(double x) {
41     return sin(2.0 * x);
42 }
```

Függvény határozott integráljának kiszámítása [3/5]

fgvint.c [44–67]

```
44 double help(tablaelem_t *tp) {
45     printf("Kérem␣válasszon␣a␣következő␣függvények␣közül:␣");
46     for (; tp -> nev; ++tp) {
47         printf("%s␣", tp -> nev);
48     }
49     printf("\nMeg␣kell␣adnia␣az␣intervallumot␣és␣a␣finomságot.\n");
50     exit(EXIT_SUCCESS);
51 }
52                                         /* Az integrálandó függvény típusa */
53 typedef double (*fuggveny_t)(double x);
54
55 static double trapez(fuggveny_t f,                  /* f(x)-t integráljuk az */
56                      double a, double b,              /* a,b intervallumon */
57                      int n) {           /* n részre osztva az intervallumot */
58                     /* Közelítő integrálás a trapéz szabály segítségével. */
59     const double h = (b - a) / n;
60     double area = 0.0;
61     int i;                                                /* a ciklusváltozó */
62
63     for (i = 1; i < n; ++i) {                     /* fgv. értékek összegzése */
64         area += (*f)(a + i * h);
65     }
66     return (area * h + ((*f)(a) + (*f)(b)) / 2.0 * h);
67 }
```

Függvény határozott integráljának kiszámítása [4/5]

fgvint.c [69–93]

```
69 int main(int argc, char **argv) {
70     tablaelem_t *tp;
71     double a, b;
72     int n;
73     char *kiiratas_formatuma = "\n␣%s(%s)=␣%g\n␣%s(%s)=␣%g\n␣%s␣integrálja=␣%g\n";
74 
75     if (argc > 5) {
76         errno = E2BIG;
77         perror("A␣hiba␣az,␣hogy");
78         exit(EXIT_FAILURE);
79     }
80 
81     for (tp = tablazat;                          /* a függvény nevét keressük */
82           (tp -> nev && argv[1] && strcmp(tp -> nev, argv[1]));
83           tp++
84         );                                                /* üres ciklusmag */
85 
86     if (tp -> nev == NULL) {                             /* nem találtuk meg */
87         printf("%s␣függvény␣még␣nincs␣megvalósítva\n", argv[1]);
88         exit(EXIT_SUCCESS);
89     }
90 
91     if (tp -> fuggveny == help) {
92         help(tablazat);
93     }
```

Függvény határozott integráljának kiszámítása [5/5]

fgvint.c [95–114]

```
95      if (argc < 5) {
96          perror("A hiba az, hogy kevés az argumentum");
97          exit(EXIT_FAILURE);
98      }
99
100     a = atof(argv[2]);                    /* Konvertál sztring-ről double-re */
101     b = atof(argv[3]);
102     n = atoi(argv[4]);                     /* Konvertál string-ről int-re */
103     printf(kiiratas_formatuma,            /* Pointer a formátum sztringre */
104             argv[1],                        /* Pointer a függvény nevére */
105             argv[2],                           /* Pointer az első számra */
106             (*tp -> fuggveny)(a),             /* A függvény értéke a-nál */
107             argv[1],                        /* Pointer a függvény nevére */
108             argv[3],                          /* Pointer a második számra */
109             (*tp -> fuggveny)(b),             /* A függvény értéke b-nél */
110             argv[1],                        /* Pointer a függvény nevére */
111             trapez(*tp -> fuggveny, a, b, n)       /* A függvény integrálja */
112         );
113     exit(EXIT_SUCCESS);
114 }
```

Flexibilis tömb egyszerű megvalósítás [1/3]

ftomb_simple_t.h [1–20]

```
1#ifndef FTOMB_SIMPLE_H
2#define FTOMB_SIMPLE_H
3
4typedef ???          elem_t;         /* a tömb elemtípusa */
5typedef unsigned int index_t;
6typedef struct flex_tomb_t {
7        elem_t *tomb;                                /* tömb */
8        index_t hatar;             /* aktuális indexhatár */
9} flex_tomb_t;
10
11/* A műveletek deklarációja:                                */
12void kiolvas(flex_tomb_t a, index_t i, elem_t *x);
13void modosit(flex_tomb_t a, index_t i, elem_t x);
14index_t felso(flex_tomb_t a);
15void letesit(flex_tomb_t *a, index_t n);
16void megszuntet(flex_tomb_t *a);
17void novel(flex_tomb_t *a, index_t d);
18void csokkent(flex_tomb_t *a, index_t d);
19
20#endif // FTOMB_SIMPLE_H
```

Flexibilis tömb egyszerű megvalósítás [2/3]

ftomb_simple_t.c [1–18]

```c
#include <stdlib.h>
#include "ftomb_simple_t.h"

void kiolvas(flex_tomb_t a, index_t i, elem_t *x) {
    if (i < a.hatar) {
        *x = a.tomb[i];
    }
}

void modosit(flex_tomb_t a, index_t i, elem_t x) {
    if (i < a.hatar) {
        a.tomb[i] = x;
    }
}

index_t felso(flex_tomb_t a) {
    return a.hatar;
}
```

Flexibilis tömb egyszerű megvalósítás [3/3]

ftomb_simple_t.c [20–37]

```
20 void letesit(flex_tomb_t *a, index_t n) {
21     a->hatar = n;
22     a->tomb = (elem_t*) ((n) ? malloc(n * sizeof(elem_t)) : NULL);
23 }
24
25 void megszuntet(flex_tomb_t *a) {
26     free(a->tomb);          /* A free() függvény jól kezeli a NULL értéket */
27     a->tomb = NULL;
28     a->hatar = 0;
29 }
30
31 void novel(flex_tomb_t *a, index_t d) {
32     a->tomb = (elem_t*)realloc(a->tomb, (a->hatar += d) * sizeof(elem_t));
33 }
34
35 void csokkent(flex_tomb_t *a, index_t d) {
36     a->tomb = (elem_t*)realloc(a->tomb, (a->hatar -= d) * sizeof(elem_t));
37 }
```

Flexibilis tömb megvalósítás [1/4]

ftomb_t.h [1–24]

```
1#ifndef FTOMB_H
2#define FTOMB_H
3
4#define L ???                                   /* lapméret */
5
6typedef ???           elem_t;          /* a tömb elemtípusa */
7typedef unsigned int index_t;
8typedef elem_t *lap_t;
9typedef lap_t  *lapterkep_t;
10typedef struct flex_tomb_t {
11        lapterkep_t lt;                        /* laptérkép */
12        index_t     hatar;          /* aktuális indexhatár */
13} flex_tomb_t;
14
15/* A műveletek deklarációja:                              */
16void kiolvas(flex_tomb_t a, index_t i, elem_t *x);
17void modosit(flex_tomb_t a, index_t i, elem_t x);
18index_t felso(flex_tomb_t a);
19void letesit(flex_tomb_t *a, index_t n);
20void megszuntet(flex_tomb_t *a);
21void novel(flex_tomb_t *a, index_t d);
22void csokkent(flex_tomb_t *a, index_t d);
23
24#endif // FTOMB_H
```

Flexibilis tömb megvalósítás [2/4]

ftomb_t.c [1–18]

```c
#include <stdlib.h>
#include "ftomb_t.h"

void kiolvas(flex_tomb_t a, index_t i, elem_t *x) {
    if (i < a.hatar) {
        *x = a.lt[i / L][i % L];
    }
}

void modosit(flex_tomb_t a, index_t i, elem_t x) {
    if (i < a.hatar) {
        a.lt[i / L][i % L] = x;
    }
}

index_t felso(flex_tomb_t a) {
    return a.hatar;
}
```

Flexibilis tömb megvalósítás [3/4]

ftomb_t.c [20–43]

```
20 void letesit(flex_tomb_t *a, index_t n) {
21     a->hatar = n;
22     if (n) {
23         int j;
24         a->lt = (elem_t**) malloc((1 + ((n - 1) / L)) * sizeof(lap_t));
25         for (j = 0; j <= ((n - 1) / L); ++j) {                /* lapok létesítése */
26             a->lt[j] = (elem_t*) malloc(L * sizeof(elem_t));
27         }
28     } else {
29         a->lt = NULL;
30     }
31 }
32
33 void megszuntet(flex_tomb_t *a) {
34     if (a->hatar) {
35         int j;
36         for (j = 0; j <= ((a->hatar - 1) / L); ++j) {            /* lapok törlése */
37             free(a->lt[j]);
38         }
39         free(a->lt);
40         a->lt = NULL;
41         a->hatar = 0;
42     }
43 }
```

Flexibilis tömb megvalósítás [4/4]

ftomb_t.c [45–66]

```
45 void novel(flex_tomb_t *a, index_t d) {
46     int j;
47     a->lt = (elem_t**) realloc(a->lt,
48                                (1 + ((a->hatar + d - 1) / L)) * sizeof(lap_t));
49     for (j = ((a->hatar) ? ((a->hatar - 1) / L) + 1 : 0);           /*   új lapok */
50                                j <= (a->hatar + d - 1) / L; ++j) {     /* létesítése */
51         a->lt[j] = (elem_t*) malloc(L * sizeof(elem_t));
52     }
53     a->hatar += d;
54 }
55
56 void csokkent(flex_tomb_t *a, index_t d) {
57     if (d <= a->hatar) {
58         int j;
59         for (j = (a->hatar - d - 1) / L + 1; j <= (a->hatar - 1) / L; ++j) {
60             free(a->lt[j]);                              /* felesleges lapok törlése */
61         }
62         a->hatar -= d;
63         a->lt = (elem_t**) realloc(a->lt,
64                                (1 + ((a->hatar - 1) / L)) * sizeof(lap_t));
65     }
66 }
```

Gráf interfész [1/1]

graf.h [1–27]

```
1  /* Gráf megvalósítása típuselrejtéssel. Közös header fájl.
2   * 2006. Augusztus 17. Gergely Tamás, gertom.inf.u-szeged.hu
3   * 2017. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
4   */
5
6  #ifndef GRAF_H
7  #define GRAF_H
8
9  #include <stdbool.h>
10
11 /* Típusdefiníciók */
12
13 typedef void *graf_t;
14 typedef int pont_t;
15
16 graf_t gr_letesit(int);
17 int  gr_pontok_szama(graf_t);
18 void gr_el_beszur(graf_t, pont_t, pont_t);
19 void gr_el_torol(graf_t, pont_t, pont_t);
20 bool gr_van_e_el(graf_t, pont_t, pont_t);
21 int  gr_pont_ki_fok(graf_t, pont_t);
22 int  gr_pont_ki_elek(graf_t, pont_t, pont_t[]);
23 int  gr_pont_be_fok(graf_t, pont_t);
24 int  gr_pont_be_elek(graf_t, pont_t, pont_t[]);
25 void gr_megszuntet(graf_t);
26
27 #endif /* GRAF_H */
```

Gráf megvalósítás 1. verzió (mátrix) [1/6]

graf1.c [1–16]

```c
/* Gráf megvalósítása típuselrejtéssel.
 * 2006. Augusztus 17. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2017. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdlib.h>
#include "graf.h"

typedef struct _graf_t {
    int   n;
    char *mx;
} _graf_t;

static bool jo_pont(_graf_t *g, pont_t p) {
    return 0 <= p && p < g->n;
}
```

Gráf megvalósítás 1. verzió (mátrix) [2/6]

graf1.c [18–39]

```
18 graf_t gr_letesit(int n) {
19     _graf_t *ptr;
20     ptr = (_graf_t*)malloc(sizeof(_graf_t));
21     if (ptr) {
22         ptr->n = n;
23         ptr->mx = (char*)malloc(n * n * sizeof(char));
24         if (!ptr->mx) {
25             free(ptr);
26             ptr = NULL;
27         } else {
28             int i, nn;
29             for (i = 0, nn = n * n; i < nn; ++i) {
30                 ptr->mx[i] = 0;
31             }
32         }
33     }
34     return (void*)ptr;
35 }
36
37 int gr_pontok_szama(graf_t g) {
38     return ((_graf_t*)g)->n;
39 }
```

Gráf megvalósítás 1. verzió (mátrix) [3/6]

graf1.c [41–60]

```
41 void gr_el_beszur(graf_t g, pont_t f, pont_t t) {
42     _graf_t *gg = (_graf_t*)g;
43     if (jo_pont(gg, f) && jo_pont(gg, t)) {
44         gg->mx[gg->n * f + t] = 1;
45     }
46 }
47
48 void gr_el_torol(graf_t g, pont_t f, pont_t t) {
49     _graf_t *gg = (_graf_t*)g;
50     if (jo_pont(gg, f) && jo_pont(gg, t)) {
51         gg->mx[gg->n * f + t] = 0;
52     }
53 }
54
55 bool gr_van_e_el(graf_t g, pont_t f, pont_t t) {
56     _graf_t *gg = (_graf_t*)g;
57     return jo_pont(gg, f) &&
58            jo_pont(gg, t) &&
59            gg->mx[gg->n * f + t];
60 }
```

Gráf megvalósítás 1. verzió (mátrix) [4/6]

graf1.c [62–84]

```
62 int gr_pont_ki_fok(graf_t g, pont_t p) {
63     int i, fok = -1;
64     _graf_t *gg = (_graf_t*)g;
65     if (jo_pont(gg, p)) {
66         for (fok = i = 0; i < gg->n; ++i) {
67             fok += gg->mx[gg->n * p + i];
68         }
69     }
70     return fok;
71 }
72
73 int gr_pont_ki_elek(graf_t g, pont_t p, pont_t l[]) {
74     int i, fok = -1;
75     _graf_t *gg = (_graf_t*)g;
76     if (jo_pont(gg, p)) {
77         for (fok = i = 0; i < gg->n; ++i) {
78             if (gg->mx[gg->n * p + i]) {
79                 l[fok++] = i;
80             }
81         }
82     }
83     return fok;
84 }
```

Gráf megvalósítás 1. verzió (mátrix) [5/6]

graf1.c [86–108]

```
86 int gr_pont_be_fok(graf_t g, pont_t p) {
87     int i, fok = -1;
88     _graf_t *gg = (_graf_t*)g;
89     if (jo_pont(gg, p)) {
90         for (fok = i = 0; i < gg->n; ++i) {
91             fok += gg->mx[gg->n * i + p];
92         }
93     }
94     return fok;
95 }
96
97 int gr_pont_be_elek(graf_t g, pont_t p, pont_t l[]) {
98     int i, fok = -1;
99     _graf_t *gg = (_graf_t*)g;
100    if (jo_pont(gg, p)) {
101        for (fok = i = 0; i < gg->n; ++i) {
102            if (gg->mx[gg->n * i + p]) {
103                l[fok++] = i;
104            }
105        }
106    }
107    return fok;
108 }
```

Gráf megvalósítás 1. verzió (mátrix) [6/6]

graf1.c [110–116]

```
110 void gr_megszuntet(graf_t g) {
111     _graf_t *gg = (_graf_t*)g;
112     if (gg) {
113         free(gg->mx);
114     }
115     free(gg);
116 }
```

Gráf megvalósítás 2. verzió (éllista) [1/6]

graf2.c [1–18]

```c
/* Gráf megvalósítása típuselrejtéssel.
 * 2006. Augusztus 17. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2017. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdlib.h>
#include "graf.h"

typedef struct _graf_t {
    int    n;
    int   *be;
    int   *ki;
    pont_t *mx;
} _graf_t;

static bool jo_pont(_graf_t *g, pont_t p) {
    return 0 <= p && p < g->n;
}
```

Gráf megvalósítás 2. verzió (éllista) [2/6]

graf2.c [20–44]

```
20 graf_t gr_letesit(int n) {
21     _graf_t *ptr;
22     ptr = (_graf_t*)malloc(sizeof(_graf_t));
23     if (ptr) {
24         ptr->n =n;
25         ptr->be = (int*)malloc(n * sizeof(int));
26         ptr->ki = (int*)malloc(n * sizeof(int));
27         ptr->mx = (pont_t*)malloc(n * n * sizeof(pont_t));
28         if (!(ptr->mx && ptr->be && ptr->ki)) {
29             free(ptr->mx); free(ptr->be); free(ptr->ki);
30             free(ptr); ptr = NULL;
31         } else {
32             int i;
33             for (i = 0; i < n; ++i) {
34                 ptr->be[i] = 0;
35                 ptr->ki[i] = 0;
36             }
37         }
38     }
39     return (void*)ptr;
40 }
41
42 int gr_pontok_szama(graf_t g) {
43     return ((_graf_t*)g)->n;
44 }
```

Gráf megvalósítás 2. verzió (éllista) [3/6]

graf2.c [46–67]

```
46 void gr_el_beszur(graf_t g, pont_t f, pont_t t) {
47     _graf_t *gg = (_graf_t*)g;
48     if (jo_pont(gg, f) && jo_pont(gg, t) && !gr_van_e_el(g, f, t)) {
49         gg->mx[gg->n * f + gg->ki[f]++] = t;
50         ++(gg->be[t]);
51     }
52 }
53
54 void gr_el_torol(graf_t g, pont_t f, pont_t t) {
55     int i;
56     _graf_t *gg = (_graf_t*)g;
57     if (jo_pont(gg, f) && jo_pont(gg, t)) {
58         for (i = 0; (i < gg->ki[f]) && (gg->mx[gg->n * f + i] != t); ++i);
59         if (i < gg->ki[f]) {
60             for (; (i < gg->ki[f]); ++i) {
61                 gg->mx[gg->n * f + i] = gg->mx[gg->n * f + i + 1];
62             }
63             --(gg->ki[f]);
64             --(gg->be[t]);
65         }
66     }
67 }
```

Gráf megvalósítás 2. verzió (éllista) [4/6]

graf2.c [69–85]

```
69 bool gr_van_e_el(graf_t g, pont_t f, pont_t t) {
70     int i;
71     _graf_t *gg = (_graf_t*)g;
72     if (jo_pont(gg, f) && jo_pont(gg, t)) {
73         for (i = 0; (i < gg->ki[f]) && (gg->mx[gg->n * f + i] != t); ++i);
74         return i < gg->ki[f];
75     }
76     return false;
77 }
78
79 int gr_pont_ki_fok(graf_t g, pont_t p) {
80     _graf_t *gg = (_graf_t*)g;
81     if (jo_pont(gg, p)) {
82         return gg->ki[p];
83     }
84     return -1;
85 }
```

Gráf megvalósítás 2. verzió (éllista) [5/6]

graf2.c [87–105]

```
87 int gr_pont_ki_elek(graf_t g, pont_t p, pont_t l[]) {
88     int i, *ptr;
89     _graf_t *gg = (_graf_t*)g;
90     if (jo_pont(gg, p)) {
91         for (i = 0, ptr = gg->mx + p * gg->n + i; i < gg->ki[p]; ++i, ++ptr) {
92             l[i] = *ptr;
93         }
94         return gg->ki[p];
95     }
96     return -1;
97 }
98
99 int gr_pont_be_fok(graf_t g, pont_t p) {
100     _graf_t *gg = (_graf_t*)g;
101     if (jo_pont(gg, p)) {
102         return gg->be[p];
103     }
104     return -1;
105 }
```

Gráf megvalósítás 2. verzió (éllista) [6/6]

graf2.c [107–127]

```
107 int gr_pont_be_elek(graf_t g, pont_t p, pont_t l[]) {
108     int i, *ptr;
109     _graf_t *gg = (_graf_t*)g;
110     if (jo_pont(gg, p)) {
111         for (i = 0, ptr = gg->mx + p * gg->n + i; i < gg->ki[p]; ++i, ++ptr) {
112             l[i] = *ptr;
113         }
114         return gg->be[p];
115     }
116     return -1;
117 }
118
119 void gr_megszuntet(graf_t g) {
120     _graf_t *gg = (_graf_t*)g;
121     if (gg) {
122         free(gg->mx);
123         free(gg->be);
124         free(gg->ki);
125     }
126     free(gg);
127 }
```

main függvény a gráf modulokhoz [1/3]

grafmain.c [1–27]

```
1  /* Gráf megvalósítása típuselrejtéssel. Főprogram.
2   * 2006. Augusztus 17. Gergely Tamás, gertom.inf.u-szeged.hu
3   */
4
5  #include <stdio.h>
6  #include <time.h>
7  #include "graf.h"
8
9  #define ISMBEGIN(s,c,I) printf("%30s (*%8d) ", s, I); \
10                 START(c) { int _ism; for (_ism = 0; _ism < I; ++_ism) {
11 #define ISMEND(cl,cg)    } PRINT(cl,cg) }
12 #define ISM_A       50000000
13 #define ISM_B       20000000
14 #define ISM_C       10000000
15 #define ISM_D        5000000
16 #define ISM_E        2000000
17 #define ISM_F        1000000
18 #define ISM_G         500000
19 #define ISM_H         200000
20 #define ISM_I         100000
21 #define ISM_J          50000
22 #define ISM_K          20000
23 #define ISM_L          10000
24 #define ELEK             100
25 #define START(c)         c = time(NULL);
26 #define PRINT(cl,cg)     { long now = time(NULL); \
27                 printf("Eltelt idő: %3ld (%3ld) másodperc\n", now - cl, now - cg); }
```

main függvény a gráf modulokhoz [2/3]

grafmain.c [29–55]

```
29 int main() {
30     graf_t G1;
31     time_t glob, loc;
32     pont_t i, j, pt[ELEK];
33 
34     G1 = gr_letesit(ELEK);
35     START(glob)
36     ISMBEGIN("Élek␣beszúrása.",loc,ISM_H)
37         for (i = 1; i < ELEK; ++i) {
38             for (j = i; j < ELEK; j += i) {
39                 gr_el_beszur(G1, i, j);
40             }
41         }
42     ISMEND(loc, glob)
43     ISMBEGIN("␣Van-e␣él␣két␣pont␣között?",loc,ISM_K)
44         for (i = 0; i < ELEK; ++i) {
45             for (j = 0; j < ELEK; ++j) {
46                 gr_van_e_el(G1, i, j);
47             }
48         }
49     ISMEND(loc, glob)
50     ISMBEGIN("Kimenő␣élek␣az␣1.␣pontból.",loc,ISM_A)
51         gr_pont_ki_elek(G1, 1, pt);
52     ISMEND(loc, glob)
53     ISMBEGIN("Kimenő␣élek␣az␣2.␣pontból.",loc,ISM_A)
54         gr_pont_ki_elek(G1, 2, pt);
55     ISMEND(loc, glob)
```

main függvény a gráf modulokhoz [3/3]

grafmain.c [56–79]

```
56      ISMBEGIN("Kimenő␣élek␣a␣3.␣pontból.",loc,ISM_A)
57          gr_pont_ki_elek(G1, 3, pt);
58      ISMEND(loc, glob)
59      ISMBEGIN("Kimenő␣élek␣a␣10.␣pontból.",loc,ISM_A)
60          gr_pont_ki_elek(G1, 10, pt);
61      ISMEND(loc, glob)
62      ISMBEGIN("Kimenő␣élek␣a␣90.␣pontból.",loc,ISM_A)
63          gr_pont_ki_elek(G1, 90, pt);
64      ISMEND(loc, glob)
65      ISMBEGIN("Fokszámok␣meghatározása.",loc,ISM_F)
66          for (i = 0; i < ELEK; ++i) {
67              gr_pont_be_fok(G1, i);
68          }
69      ISMEND(loc,glob)
70      ISMBEGIN("Élek␣törlése.",loc,ISM_H)
71          for (i = 1; i < ELEK; ++i) {
72              for (j = i; j < ELEK; j += i) {
73                  gr_el_torol(G1, i, j);
74              }
75          }
76      ISMEND(loc,glob)
77      gr_megszuntet(G1);
78      return 0;
79  }
```

Halmaz modul interfész [1/2]

halmaz.h [1–23]

```
1  /* Header fájl a halmaz modulhoz. A logikai típust most a
2   * preprocesszor segítségével definiáljuk.
3   * 2006. Augusztus 16. Gergely Tamás, gertom@inf.u-szeged.hu
4   * 2014. Március 31. Gergely Tamás, gertom@inf.u-szeged.hu
5   */
6
7  #ifndef HALMAZ_H
8  #define HALMAZ_H
9
10 #include <stdbool.h>
11
12 #define K (8 * sizeof(kishalmaz_t))
13 #define HALMAZELEM_PRINT_FORMAT "lld"
14
15 typedef long long int kishalmaz_t;
16
17 typedef long long int halmazelem_t;
18
19 typedef struct {
20         halmazelem_t  n;              /* Az univerzum a [0..N) */
21         long long int m;                /* A kishalmazok száma */
22         kishalmaz_t  *tar;             /* A kishalmazok tömbje */
23 } halmaz_t;
```

Halmaz modul interfész [2/2]

halmaz.h [25–39]

```
25 halmaz_t letesit(halmazelem_t);
26 void megszuntet(halmaz_t);
27 void uresit(halmaz_t);
28 void bovit(halmaz_t, halmazelem_t);
29 void torol(halmaz_t, halmazelem_t);
30 bool eleme(halmaz_t, halmazelem_t);
31 void egyesites(halmaz_t, halmaz_t, halmaz_t);
32 void metszet(halmaz_t, halmaz_t, halmaz_t);
33 void kulonbseg(halmaz_t, halmaz_t, halmaz_t);
34 bool egyenlo(halmaz_t, halmaz_t);
35 bool resz(halmaz_t, halmaz_t);
36 bool ures_e(halmaz_t);
37 void ertekadas(halmaz_t, halmaz_t);
38
39 #endif /* HALMAZ_H */
```

Halmaz modul implementáció [1/4]

halmaz.c [1–25]

```c
/* A halmaz modul függvényeinek megvalósítása.
 * 2006. Augusztus 16. Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdlib.h>
#include "halmaz.h"

halmaz_t letesit(halmazelem_t n) {
    halmaz_t ret;
    ret.n = n;
    if (n) {
        ret.m   = (n - 1) / K + 1;
        ret.tar = (kishalmaz_t*)malloc(ret.m * sizeof(kishalmaz_t));
    } else {
        ret.m   = 0;
        ret.tar = NULL;
    }
    return ret;
}

void megszuntet(halmaz_t h) {
    free(h.tar);
    h.n = h.m = 0;
    h.tar = NULL;
}
```

Halmaz modul implementáció [2/4]

halmaz.c [27–49]

```
27 void uresit(halmaz_t h) {
28     long long int i;
29     for (i = 0; i < h.m; ++i) {
30         h.tar[i] = 0;
31     }
32 }
33
34 void bovit(halmaz_t h, halmazelem_t x) {
35     if (x < h.n) {
36         h.tar[x / K] |= (1LL << (x % K));
37     }
38 }
39
40 void torol(halmaz_t h, halmazelem_t x) {
41     if (x < h.n) {
42         h.tar[x / K] &= ~(1LL << (x % K));
43     }
44 }
45
46 bool eleme(halmaz_t h, halmazelem_t x) {
47     return (x < h.n) &&
48            (h.tar[x / K] & (1LL << (x % K)));
49 }
```

Halmaz modul implementáció [3/4]

halmaz.c [51–70]

```
51 void egyesites(halmaz_t lhs, halmaz_t rhs, halmaz_t dst) {
52     long long int i;
53     for (i = 0; i < dst.m; ++i) {
54         dst.tar[i] = lhs.tar[i] | rhs.tar[i];
55     }
56 }
57
58 void metszet(halmaz_t lhs, halmaz_t rhs, halmaz_t dst) {
59     long long int i;
60     for (i = 0; i < dst.m; ++i) {
61         dst.tar[i] = lhs.tar[i] & rhs.tar[i];
62     }
63 }
64
65 void kulonbseg(halmaz_t lhs, halmaz_t rhs, halmaz_t dst) {
66     long long int i;
67     for (i = 0; i < dst.m; ++i) {
68         dst.tar[i] = lhs.tar[i] & ~(rhs.tar[i]);
69     }
70 }
```

Halmaz modul implementáció [4/4]

halmaz.c [72–95]

```
72 bool egyenlo(halmaz_t lhs, halmaz_t rhs) {
73     long long int i;
74     for (i = 0; (i < lhs.m) && (lhs.tar[i] == rhs.tar[i]); ++i);
75     return i == lhs.m;
76 }
77
78 bool resz(halmaz_t lhs, halmaz_t rhs) {
79     long long int i;
80     for (i = 0; (i < lhs.m) && !(lhs.tar[i] & ~(rhs.tar[i])); ++i);
81     return i == lhs.m;
82 }
83
84 bool ures_e(halmaz_t h) {
85     long long int i;
86     for (i = 0; (i < h.m) && !(h.tar[i]); ++i);
87     return i == h.m;
88 }
89
90 void ertekadas(halmaz_t dst, halmaz_t src) {
91     long long int i;
92     for (i = 0; i < dst.m; ++i) {
93         dst.tar[i] = src.tar[i];
94     }
95 }
```

Halmaz megvalósítás [1/4]

halmaz_t.h [1–26]

```
1#ifndef HALMAZ_H
2#define HALMAZ_H
3
4#include <stdbool.h>
5
6#define K (8*sizeof(KisHalmaz))
7#define M ???
8#define H_MAX (M*K)
9
10typedef unsigned long int halmazelem;
11typedef halmazelem KisHalmaz;
12typedef KisHalmaz Halmaz[M];
13
14void Uresit(Halmaz);
15void Bovit(Halmaz, halmazelem);
16void Torol(Halmaz, halmazelem);
17bool Eleme(Halmaz, halmazelem);
18void Egyesites(Halmaz H1, Halmaz H2, Halmaz H);
19void Metszet(Halmaz H1, Halmaz H2, Halmaz H);
20void Kulonbseg(Halmaz H1, Halmaz H2, Halmaz H);
21bool Egyenlo(Halmaz H1, Halmaz H2);
22bool Resz(Halmaz H1, Halmaz H2);
23bool Urese(Halmaz H);
24void Ertekadas(Halmaz H1, Halmaz H2);
25
26#endif // HALMAZ_H
```

Halmaz megvalósítás [2/4]

halmaz_t.c [1–21]

```
1#include "halmaz_t.h"
2
3void Uresit(Halmaz H) {
4    long int i;
5    for (i = 0; i < M; ++i)
6        H[i] = 0;
7}
8
9void Bovit(Halmaz H, halmazelem x) {
10    if (x < H_MAX)
11        H[x / K] |=  (1l << (x % K));
12}
13
14void Torol(Halmaz H, halmazelem x) {
15    if (x < H_MAX)
16        H[x / K] &= ~(1l << (x % K));
17}
18
19bool Eleme(halmazelem x, Halmaz H) {
20    return (x < H_MAX) && (H[x / K] & (1l << (x % K)));
21}
```

Halmaz megvalósítás [3/4]

halmaz_t.c [23–45]

```
23 void Egyesites(Halmaz H1, Halmaz H2, Halmaz H) {
24     long int i;
25     for (i = 0; i < M; ++i)
26         H[i] = H1[i] | H2[i];
27 }
28
29 void Metszet(Halmaz H1, Halmaz H2, Halmaz H) {
30     long int i;
31     for (i = 0; i < M; ++i)
32         H[i] = H1[i] & H2[i];
33 }
34
35 void Kulonbseg(Halmaz H1, Halmaz H2, Halmaz H) {
36     long int i;
37     for (i = 0; i < M; ++i)
38         H[i] = H1[i] & ~(H2[i]);
39 }
40
41 bool Egyenlo(Halmaz H1, Halmaz H2) {
42     long int i;
43     for (i = 0; (i < M) && (H1[i] == H2[i]); ++i);
44     return i == M;
45 }
```

Halmaz megvalósítás [4/4]

halmaz_t.c [47–63]

```
47 bool Resz(Halmaz H1, Halmaz H2) {
48     long int i;
49     for (i = 0; (i<M) && !(H1[i] & ~(H2[i])); ++i);
50     return i == M;
51 }
52
53 bool Urese(Halmaz H) {
54     long int i;
55     for (i = 0; (i < M) && !(H[i]); ++i);
56     return i == M;
57 }
58
59 void Ertekadas(Halmaz H1, Halmaz H2) {
60     long int i;
61     for (i = 0; i < M; ++i)
62         H1[i] = H2[i];
63 }
```

Hanoi tornyai [1/2]

hanoi.c [1–21]

```c
/* A Hanoi tornyai játék megvalósítása rekurzív eljárással.
 * 1997. Október 31.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

void mozgat(int innen, int ide) {          /* Átrak egy korongot innen ide */
    printf("␣Tegyünk␣át␣egy␣korongot␣a␣%d.␣oszlopról␣a␣%d.␣oszlopra!\n", innen, ide);
}

void hanoi(int n,                                   /* ilyen magas a torony */
           int rol,                                    /* erről a toronyról */
           int ra) {                                     /* erre a toronyra */
    if (n == 1) {
        mozgat(rol, ra);
    } else {
        hanoi(n - 1, rol, 6 - ra - rol);
        mozgat(rol, ra);
        hanoi(n - 1, 6 - ra - rol, ra);
    }
}
```

Hanoi tornyai [2/2]

hanoi.c [23–42]

```
23 int main() {
24     int honnan;                          /* erről a toronyról kell átrakni */
25     int hova;                                      /* erre a toronyra */
26     int db;                              /* a torony ennyi korongból áll */
27
28     printf("Kérem␣adja␣meg␣a␣torony␣magasságát:␣");
29     scanf("%d%*[^\n]", &db); getchar();
30     printf("Kérem␣adja␣meg,␣hogy␣a␣torony␣hol␣áll?␣(1,2,3)␣");
31     scanf("%d%*[^\n]", &honnan); getchar();
32     printf("Kérem␣adja␣meg,␣hogy␣melyik␣oszlopra␣tegyük␣át?␣");
33     scanf("%d%*[^\n]", &hova); getchar();
34     if (db > 0 &&
35         1 <= honnan && honnan <= 3 && 1 <= hova && hova <= 3 &&
36         honnan != hova) {
37         hanoi(db, honnan, hova);
38     } else {
39         printf("Hibás␣adat\n");
40     }
41     return 0;
42 }
```

Háromszög osztályozása [1/2]

haromszog.c [1–25]

```c
/* Milyen háromszöget határoz meg három pozitiv valós szám,
 *    mint a háromszög három oldalhosszúsága?
 * 1997. Október 13.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double a, b, c;     /* a háromszög oldalhosszúságai */
    double m;           /* munkaváltozó a cseréhez      */

    printf("Kérem a három pozitív valós számot!\n");
    scanf("%lf%lf%lf", &a, &b, &c);

                              /* a,b,c átrendezése úgy, hogy a>=b,c legyen */
    if (a < b) {                                    /* a és b átrendezése */
        m = a;
        a = b;
        b = m;
    }
    if (a < c) {                                    /* a és c átrendezése */
        m = a;
        a = c;
        c = m;
    }
```

Háromszög osztályozása [2/2]

haromszog.c [27–46]

```
27      if (b <= 0 || c <= 0) {
28          printf("Nem háromszög!\n");                    /* 1. alternatíva */
29      } else if (a >= b + c) {
30          printf("Nem háromszög!\n");                    /* 2. alternatíva */
31      } else if (a == b && b == c) {
32          printf("Szabályos háromszög.\n");              /* 3. alternatíva */
33      } else if (a == b || b == c || a == c) {
34          if (a * a == b * b + c * c) {                  /* 4. alternatíva */
35              printf("Egyenlő szárú derékszögű háromszög.\n");
36          } else {
37              printf("Egyenlő szárú háromszög.\n");
38          }
39      } else if (a * a == b * b + c * c) {
40          printf("Derékszögű háromszög.\n");             /* 5. alternatíva */
41      } else {
42          printf("Egyéb háromszög.\n");                      /* egyébként */
43      }
44                                          /* vége a többszörös szelekciónak */
45      return 0;
46  }
```

Háromszög osztályozása, toleranciával [1/2]

haromszog-tol.c [1–29]

```c
/* Milyen háromszöget határoz meg három pozitiv valós szám,
 *    mint a háromszög három oldalhosszúsága? A valós típus
 *    pontatlansága miatt toleranciával számolunk.
 * 1997. Október 13.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2014. Február 19.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

#define EQUALS(X,Y) ( ( ((X) > (Y)) ? ((X) - (Y)) : ((Y) - (X)) ) <= 1e-10 )

int main() {
    double a, b, c;     /* a háromszög oldalhosszúságai */
    double m;           /* munkaváltozó a cseréhez      */

    printf("Kérem a három pozitív valós számot!\n");
    scanf("%lf%lf%lf", &a, &b, &c);

                            /* a,b,c átrendezése úgy, hogy a>=b,c legyen */
    if (a < b) {                                    /* a és b átrendezése */
        m = a;
        a = b;
        b = m;
    }
    if (a < c) {                                    /* a és c átrendezése */
        m = a;
        a = c;
        c = m;
    }
```

Háromszög osztályozása, toleranciával [2/2]

haromszog-tol.c [31–50]

```
31      if (c <= 0 || b <= 0) {
32          printf("Nem háromszög!\n");                          /* 1. alternatíva */
33      } else if (a >= b + c) {
34          printf("Nem háromszög!\n");                          /* 2. alternatíva */
35      } else if (EQUALS(a,b) && EQUALS(b,c) && EQUALS(a,c)) {
36          printf("Szabályos háromszög.\n");                    /* 3. alternatíva */
37      } else if (EQUALS(a,b) || EQUALS(b,c) || EQUALS(a,c)) {
38          if (EQUALS((a * a), (b * b + c * c))) {              /* 4. alternatíva */
39              printf("Egyenlő szárú derékszögű háromszög.\n");
40          } else {
41              printf("Egyenlő szárú háromszög.\n");
42          }
43      } else if (EQUALS((a * a), (b * b + c * c))) {
44          printf("Derékszögű háromszög.\n");                   /* 5. alternatíva */
45      } else {
46          printf("Egyéb háromszög.\n");                            /* egyébként */
47      }
48                                          /* vége a többszörös szelekciónak */
49      return 0;
50  }
```

Hello Világ! [1/1]

hello.c [1–10]

```c
/* Helló Világ!
 * 2004. November 3.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    printf("Helló␣Világ!\n");
    return 0;
}
```

Határozott integrál kiszámítása [1/3]

integral.c [1–24]

```c
/* Közelítő integrálás a trapéz szabály segítségével.
 * Az integrálandó függvényt paraméterként kapjuk.
 * 1998. Április 14.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2006. Augusztus 14. Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>
#include <math.h>
                                        /* Az integrálandó függvény típusa */
typedef double (*fuggveny_t)(double x);

static double trapez(fuggveny_t f,                  /* f(x)-t integráljuk az */
                     double a, double b,              /* a,b intervallumon */
                     int n) {                /* n részre osztva az interv.-t */
                    /* Közelítő integrálás a trapéz szabály segítségével. */
    const double h = (b - a) / n;
    double area = 0.0;
    int i;                                              /* a ciklusváltozó */

    for (i = 1; i < n; ++i) {                    /* fgv. értékek összegzése */
        area += (*f)(a + i * h);                           /* f(a + i * h) */
    }
    return (area * h + ((*f)(a) + (*f)(b)) / 2.0 * h);    /* (f(a) + f(b)) */
}
```

Határozott integrál kiszámítása [2/3]

integral.c [26–34]

```
26 static double expx(double x) {
27                                          /* az első integrálandó függvény */
28     return (exp(x) / x);
29 }
30
31 static double sin2x(double x) {
32                                        /* az második integrálandó függvény */
33     return sin(2.0 * x);
34 }
```

Határozott integrál kiszámítása [3/3]

integral.c [36–58]

```
36 int main() {
37     double a, b;
38     int n;
39
40     printf("Az␣exp(x)/x␣függvény␣közelítő␣integrálja.\n");
41     printf("Kérem␣az␣integrálási␣intervallumot␣");
42     printf("és␣az␣osztáspontok␣számát␣(a,b,n)!\n");
43     printf("a:?␣"); scanf("%lg%*[^\n]", &a); getchar();
44     printf("b:?␣"); scanf("%lg%*[^\n]", &b); getchar();
45     printf("n:?␣"); scanf("%d%*[^\n]", &n); getchar();
46     printf("Az␣integrál␣közelítő␣értéke:␣");
47     printf("%10.5f\n", trapez(expx, a, b, n));
48
49     printf("Az␣sin(2x)␣függvény␣közelítő␣integrálja.\n");
50     printf("Kérem␣az␣integrálási␣intervallumot␣");
51     printf("és␣az␣osztáspontok␣számát␣(a,b,n)!\n");
52     printf("a:?␣"); scanf("%lg%*[^\n]", &a); getchar();
53     printf("b:?␣"); scanf("%lg%*[^\n]", &b); getchar();
54     printf("n:?␣"); scanf("%d%*[^\n]", &n); getchar();
55     printf("Az␣integrál␣közelítő␣értéke:␣");
56     printf("%10.5f\n", trapez(sin2x, a, b, n));
57     return 0;
58 }
```

Kamatos kamat számítása [1/2]

kamatos.c [1–22]

```c
/* Kamatos-kamat számítás a hatványozás függvény segítségével.
 * 1997. Október 31.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2014. Szeptember 23. Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

double hatvany(double x, int n) {
                                    /* x n-edik hatványát kiszámító függvény */
    if (n == 0) {
        return 1.0;
    } else if (x == 0.0) {
        return 0.0;
    } else if (x > 0.0) {
        return exp(n * log(x));
    } else if (n & 1) {
        return -exp(n * log(-x));
    } else {
        return exp(n * log(-x));
    }
}
```

Kamatos kamat számítása [2/2]

kamatos.c [24–41]

```c
int main() {
    double osszeg, uj_osszeg;
    int kamatlab, ev;

    printf("␣A␣kamatozó␣összeg␣?␣");
    scanf("%lg%*[^\n]", &osszeg);
    getchar();
    printf("␣A␣kamatláb␣?␣");
    scanf("%d%*[^\n]", &kamatlab);
    getchar();
    printf("␣A␣kamatozási␣évek␣száma␣?␣");
    scanf("%d%*[^\n]", &ev);
    getchar();
    uj_osszeg = osszeg * hatvany(1.0 + kamatlab / 100.0, ev);
    printf("A␣kamatos␣kamattal␣növelt␣összeg:");
    printf("%10.2f\n", uj_osszeg);
    return 0;
}
```

Memóriaterület kiírása binárisan [1/2]

keszit.c [1–15]

```
1  /* Egy bináris adatállományba egész számok kiírása.
2   * Az adatállomány nevét a parancssorból kapjuk.
3   * 1998. Április 24.  Dévényi Károly, devenyi@inf.u-szeged.hu
4   */
5
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <fcntl.h>
9  #include <unistd.h>
10 #include <errno.h>
11
12 #define DBSZAM 10
13
14 long int tomb[DBSZAM];                          /* innen fogunk irni */
15 int i;                                           /* ciklusváltozó */
```

Memóriaterület kiírása binárisan [2/2]

keszit.c [17–40]

```
17 int main(int argc, char **argv) {
18     int fd;                                  /* fájlleíró = file descriptor */
19     int bytedb;                                      /* kiírt bájtok száma */
20
21     if ((fd = open(argv[1], O_WRONLY)) == -1) {
22         errno = ENOENT;
23         perror("Megnyitási hiba");
24         exit(EXIT_FAILURE);
25     }
26                                /* a tomb feltöltése jól látható értékekkel */
27     for (i = 0; i < DBSZAM; ++i) {
28         tomb[i] = i + 'a';
29     }
30
31     if ((bytedb = write(fd, tomb, DBSZAM * sizeof(long int))) == -1) {
32         perror("Írási hiba");
33         exit(EXIT_FAILURE);
34     }
35
36     close(fd);
37
38     printf("Sikerült kiírni %d darab bájtot\n", bytedb);
39     return 0;
40 }
```

Klikkek meghatározása absztrakt Halmaz típussal [1/2]

klikk-abstract.c [1–28]

```c
/* A beolvasott R relációt tartalmazó legszűkebb ekvivalencia
   reláció szerinti osztályozást határozzuk meg.
   Vázlat, nem fordítható C program.
   2006. Augusztus 14. Gergely Tamás, gertom@inf.u-szeged.hu
 */
#include <stdio.h>
#define N 10                                        /* maximális elemszám */
typedef Halmaz(U) Halmaz;                           /* EZ ÍGY NEM C !!!*/

int main() {
    Halmaz H[N];
    unsigned short i, j, ti, tj;
    for (i = 1; i <= N; ++i) {                      /* inicializálás */
        Uresit(H[i - 1]); Bovit(H[i - 1], i);
    }
    printf("Kérem␣a␣relációban␣lévő␣számpárokat!\n");
    scanf("%hd%hd%*[^\n]", &i, &j);
    getchar();
    while (i != 0) {
        for (ti = 0; !Eleme(i, H[ti]); ++ti);   /* amelyik H[ti] halmazban van i ? */
        for (tj = 0; !Eleme(j, H[tj]); ++tj);   /* amelyik H[tj] halmazban van j ? */
        if (ti != tj) {                             /* H[ti] és H[tj] összevonása */
            Egyesites(H[ti], H[tj], H[ti]);
            Uresit(H[tj]);
        }
        scanf("%hd%hd%*[^\n]", &i, &j);
        getchar();
    }
```

Klikkek meghatározása absztrakt Halmaz típussal [2/2]

klikk-abstract.c [30–40]

```
30      printf("Az␣osztályok:␣\n");
31      for (i = 1; i <= N; ++i) {                          /* az osztályok kiíratása */
32          if (!Urese(H[i - 1])) {
33              for (j = 1; j <= N; ++j) {
34                  if (Eleme(j, H[i - 1]))
35                      printf("%4hd,", j);
36              }
37              putchar('\n');
38          }
39      }
40  }
```

Klikkek meghatározása [1/2]

klikk.c [1–19]

```c
/* A beolvasott R relációt tartalmazó legszűkebb ekvivalencia
 *   reláció szerinti osztályozást határozzuk meg.
 * 1997. December 6.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

#define N  10                                   /* maximális elemszám */

typedef unsigned short int univerzum_t;                 /* az univerzum */
typedef long int halmaz_t;        /* N kicsi, ezért elegendő egy long int */

int main(int argc, char *argv[]) {
    halmaz_t h[N];
    univerzum_t i, j, ti, tj;

    for (i = 1; i <= N; ++i) {                          /* inicializálás */
        h[i - 1] = 1l << i;
    }
```

Klikkek meghatározása [2/2]

klikk.c [21–46]

```
21      printf("Kérem␣a␣relációban␣lévö␣számpárokat!\n");
22      scanf("%hd%hd%*[^\n]", &i, &j); getchar();
23      while (i != 0) {
24                              /* az i-t tartalmazó halmaz ti indexének keresése */
25          for (ti = 0; ((1l << i) & h[ti]) == 0; ++ti);
26                               /* a j-t tartalmazó halmaz tj indexének keresése */
27          for (tj = 0; ((1l << j) & h[tj]) == 0; ++tj);
28          if (ti != tj) {                          /* h[ti] és h[tj] összevonása */
29              h[ti] |= h[tj];
30              h[tj] = 0;
31          }
32          scanf("%hd%hd%*[^\n]", &i, &j); getchar();
33      }
34
35      printf("Az␣osztályok:\n");
36      for (i = 1; i <= N; ++i) {                      /* az osztályok kiíratása */
37          if (h[i - 1] != 0) {
38              for (j = 1; j <= N; ++j) {
39                  if (((1l << j) & h[i - 1]) != 0)
40                      printf("%4d,", j);
41              }
42              putchar('\n');
43          }
44      }
45      return 0;
46  }
```

Láncolt lista megvalósítás [1/4]

lanc_t.h [1–22]

```
1#ifndef LANC_H
2#define LANC_H
3
4typedef ???    elem_t;          /* a lista_t elemtípusa */
5typedef struct cella_t {
6    elem_t adat;                          /* adatelem */
7    struct cella_t *csat;          /* a következő elem  */
8} cella_t;
9typedef struct cella_t *pozicio_t;
10typedef pozicio_t lista_t;
11
12/* A műveletek deklarációja:                          */
13void letesit(lista_t *l);
14void megszuntet(lista_t *l);
15void kiolvas(pozicio_t p, elem_t *x);
16void modosit(pozicio_t p, elem_t x);
17void beszur(lista_t *l, pozicio_t p, elem_t x);
18void torol(lista_t *l, pozicio_t p);
19void kovetkezo(pozicio_t p, pozicio_t *n);
20void ertekadas(lista_t *lhs, lista_t rhs);
21
22#endif // LANC_H
```

Láncolt lista megvalósítás [2/4]

lanc_t.c [1–23]

```
1#include <stdlib.h>
2#include "lanc_t.h"
3
4void letesit(lista_t *l) {
5    *l = NULL;
6}
7
8void megszuntet(lista_t *l) {
9    pozicio_t p = *l;
10   while (p) {
11       p = p->csat;
12       free(*l);
13       *l = p;
14   }
15}
16
17void kiolvas(pozicio_t p, elem_t *x) {
18   *x = p->adat;
19}
20
21void modosit(pozicio_t p, elem_t x) {
22   p->adat = x;
23}
```

Láncolt lista megvalósítás [3/4]

lanc_t.c [25–51]

```
25 void beszur(lista_t *l, pozicio_t p, elem_t x) {
26     pozicio_t q = *l;
27     if (q == p) {
28         q = *l = malloc(sizeof(cella_t));
29     } else {
30         while (q->csat != p) {
31             q = q->csat;
32         }
33         q->csat = malloc(sizeof(cella_t));
34         q = q->csat;
35     }
36     q->csat = p;
37     q->adat = x;
38 }
39
40 void torol(lista_t *l, pozicio_t p) {
41     pozicio_t q = *l;
42     if (q == p) {
43         *l = p->csat;
44     } else {
45         while (q->csat != p) {
46             q = q->csat;
47         }
48         q->csat = p->csat;
49     }
50     free(p);
51 }
```

Láncolt lista megvalósítás [4/4]

lanc_t.c [53–71]

```
53 void kovetkezo(pozicio_t p, pozicio_t *n) {
54     *n = p->csat;
55 }
56
57 void ertekadas(lista_t *lhs, lista_t rhs) {
58     megszuntet(lhs);
59     if (!rhs) {
60         *lhs = NULL;
61         return;
62     }
63     pozicio_t p;
64     *lhs = p = malloc(sizeof(cella_t));
65     p->adat = rhs->adat;
66     while ((rhs = rhs->csat)) {
67         p = p->csat = malloc(sizeof(cella_t));
68         p->adat = rhs->adat;
69     }
70     p->csat = NULL;
71 }
```

Számsorozat legnagyobb közös osztója [1/2]

lnkoszt.c [1–21]

```c
/* Pozitív egész számok legnagyobb közös osztójának meghatározása.
 *   A hurok ismétléses vezérlés megvalósítása break utasítással.
 * 1997. November 7.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2006. Augusztus 8.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

int lnko(int x, int y) {
    /* x és y legnagyobb közös osztójának meghatározása
     * Euklidesz algoritmusával.
     */
    int m;

    while (y != 0) {
        m = x % y;
        x = y;
        y = m;
    }
    return x;
}
```

Számsorozat legnagyobb közös osztója [2/2]

lnkoszt.c [23–45]

```c
int main() {
    int a, b;

    printf("A␣program␣pozitív␣egész␣számok␣legnagyobb\n");
    printf("közös␣osztóját␣számítja.\n");
    printf("Kérem␣a␣számok␣sorozatát,␣amit␣0␣zár!\n");
    printf("?␣");
    scanf("%d%*[^\n]", &a); getchar();

    while (1) {                                    /* a hurok ciklus kezdete */
        printf("?␣");
        scanf("%d%*[^\n]", &b); getchar();
        if (b == 0) {                                        /* első kijárat */
            break;
        }
        a = lnko(a, b);
        if (a == 1) {                                     /* második kijárat */
            break;
        }
    }                                                  /* a hurok ciklus vége */
    printf("␣A␣számok␣legnagyobb␣közös␣osztója:␣%d\n", a);
    return 0;
}
```

Számsorozat legnagyobb közös osztója$_1$ [1/2]

lnkoszt1.c [1–21]

```c
/* Pozitív egész számok legnagyobb közös osztójának meghatározása.
 *    A hurok ismétléses vezérlés megvalósítása ismert szerkezetekkel.
 * 1997. November 7.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2006. Augusztus 8.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

int lnko(int x, int y) {
    /* x és y legnagyobb közös osztójának meghatározása
     * Euklidesz algoritmusával.
     */
    int m;

    while (y != 0) {
        m = x % y;
        x = y;
        y = m;
    }
    return x;
}
```

Számsorozat legnagyobb közös osztója$_1$ [2/2]

lnkoszt1.c [23–49]

```
23 int main() {
24     int a, b;
25     int tovabb;                  /* logikai változó a ciklus megvalósításához */
26
27     printf("A program pozitív egész számok legnagyobb\n");
28     printf("közös osztóját számítja.\n");
29     printf("Kérem a számok sorozatát, amit 0 zár!\n");
30     printf("? ");
31     scanf("%d%*[^\n]", &a); getchar();
32
33     tovabb = !0;
34     while (tovabb) {                              /*  a hurok ciklus kezdete */
35         printf("? ");
36         scanf("%d%*[^\n]", &b); getchar();
37
38         if (b == 0) {                                        /* első kijárat */
39             tovabb = 0;
40         } else {
41             a = lnko(a, b);
42             if (a == 1) {                                  /* második kijárat */
43                 tovabb = 0;
44             }
45         }
46     }                                                  /* a hurok ciklus vége */
47     printf(" A számok legnagyobb közös osztója: %d\n", a);
48     return 0;
49 }
```

Másodfokú egyenlet gyökei [1/3]

masodfok.c [1–9]

```c
/* Másodfokú egyenlet valós gyökeinek meghatározása a
 *   megoldo függvénnyel.
 * 1998. március 31. Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2013. augusztus 29. Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

Másodfokú egyenlet gyökei [2/3]

masodfok.c [11–37]

```
11 bool megoldo(double a, double b, double c, double *x1, double *x2) {
12     bool valos = true;                                   /* van-e megoldás */
13
14     if (a == 0.0) {
15         if (b == 0.0) {                                /* az egyenlet elfajuló */
16             valos = false;
17         } else {
18             *x1 = *x2 = -(c / b);
19         }
20     } else {
21         double d;                                        /* a diszkrimináns */
22         d = b * b - 4.0 * a * c;
23         if (d < 0.0) {                                 /* nincs valós gyöke */
24             valos = false;
25         } else {
26             *x1 = (-b + sqrt(d)) / (2.0 * a);
27             *x2 = (-b - sqrt(d)) / (2.0 * a);
28             if (fabs(*x1) > fabs(*x2)) {  /* gyökök pontosabb kiszámolása */
29                 *x2 = c / (*x1 * a);
30             } else if (*x2 != 0) {
31                 *x1 = c / (*x2 * a);
32             }
33         }
34     }
35
36     return valos;
37 }
```

Másodfokú egyenlet gyökei [3/3]

masodfok.c [39–57]

```
39 int main() {
40     double a, b, c, x, y;                          /* a főprogram változói */
41
42     printf("Kérem az első egyenlet együtthatóit!\n");
43     scanf("%lg%lg%lg%*[^\n]", &a, &b, &c); getchar();
44     if (megoldo(a, b, c, &x, &y)) {
45         printf("Az egyenlet gyökei: %20.10f és %20.10f\n", x, y);
46     } else {
47         printf("Az egyenletnek nincs valós megoldása!\n");
48     }
49     printf("Kérem a második egyenlet együtthatóit!\n");
50     scanf("%lg%lg%lg%*[^\n]", &a, &b, &c); getchar();
51     if (megoldo(a, b, c, &x, &y)) {
52         printf("Az egyenlet gyökei: %20.10f és %20.10f\n", x, y);
53     } else {
54         printf("Az egyenletnek nincs valós megoldása!\n");
55     }
56     return 0;
57 }
```

Másodfokú egyenlet gyökei [megoldo függvény változat]

masodfok.c [11–37] helyett

```
11 bool megoldo(double a, double b, double c, double *x1, double *x2) {
12     bool valos = true;                                    /* van-e megoldás */
13     double mx1, mx2;                        /* munkaváltozók *x1 és *x2 helyett */
14     if (a == 0.0) {
15         if (b == 0.0) {                                  /* az egyenlet elfajuló */
16             valos = false;
17         } else {
18             mx1 = mx2 = -(c / b);
19         }
20     } else {
21         double d;                                          /* a diszkrimináns */
22         d = b * b - 4.0 * a * c;
23         if (d < 0.0) {                                   /* nincs valós gyöke */
24             valos = false;
25         } else {
26             mx1 = (-b + sqrt(d)) / (2.0 * a);
27             mx2 = (-b - sqrt(d)) / (2.0 * a);
28             if (fabs(mx1) > fabs(mx2)) {  /* gyökök pontosabb kiszámolása */
29                 mx2 = c / (mx1 * a);
30             } else if (mx2 != 0) {
31                 mx1 = c / (mx2 * a);
32             }
33         }
34     }
35     *x1 = mx1; *x2 = mx2;
36     return valos;
37 }
```

Minimális C program [1/1]

minimalis.c [1–6]

```c
/* Minimális C program, ami nem csinál semmit
 * 1998. Február 26.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

main() {
}
```

Számsorozat jellemzői [1/2]

minimax.c [1–22]

```c
/* Határozzuk meg egy valós számsorozat legkisebb
 * és legnagyobb elemét, valamint a sorozat átlagát!
 * 1997. Október 25.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double vegjel, szam, osszeg, min, max, atlag;
    int db;                                  /* az összegzett elemek száma */

    printf("Ez a program valós számsorozat minimális,\n");
    printf("maximális elemét és átlagát számolja.\n");
    printf("Az input sorozatot végjel zárja.\n");
    printf("Kérem a végjelet!\n");                          /* inicializálás */
    scanf("%lf", &vegjel);
    printf("Kérem az input számsorozatot!\n");
    printf("? ");
    scanf("%lf", &szam);
    min = max = szam;
    osszeg = 0.0;
    db = 0;
```

Számsorozat jellemzői [2/2]

minimax.c [24–46]

```
24      while (szam != vegjel) {                        /* a ciklus kezdete */
25          osszeg += szam;                                     /* összegzés */
26          ++db;                                       /* számláló növelés */
27
28          if (szam < min) {                           /* min-max számítás */
29              min = szam;
30          } else if (szam > max) {
31              max = szam;
32          }
33                                                  /* a következő szám beolvasása */
34          printf("? ");
35          scanf("%lf", &szam);
36      }                                                       /* a ciklus vége */
37
38      if (db == 0) {
39          printf("Üres számsorozat érkezett.\n");
40      } else {
41          atlag = osszeg / db;
42          printf("Minimum  = %10.3f Maximum = %10.3f\n", min, max);
43          printf("Az átlag = %10.3f\n", atlag);
44      }
45      return 0;
46  }
```

Sorozat statisztikája

minimax.c (v1.0) [1–26]

```c
/* Határozzuk meg egy valós számsorozat legkisebb
 * és legnagyobb elemét, valamint a sorozat átlagát!
 * 1997. Október 25.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double vegjel, szam, min, max, atlag = 0.0;
    int db = 0;
    printf("Kérem a végjelet: ");
    scanf("%lf", &vegjel);
    printf("Kérem a sorozatot:\n? ");
    scanf("%lf", &szam);
    min = max = szam;
    while (szam != vegjel) {
        if (szam < min) {
            min = szam;
        } else if (szam > max) {
            max = szam;
        }
        atlag += szam;
        ++db;
        printf("? ");
        scanf("%lf", &szam);
    }
```

Sorozat statisztikája

minimax.c (v1.0) [27–36]

```
27      if (db == 0) {
28          printf("Üres␣számsorozat␣érkezett.\n");
29      } else {
30          atlag /= db;
31          printf("Minimum␣␣␣=␣%10.3f\n", min);
32          printf("Maximum␣␣␣=␣%10.3f\n", max);
33          printf("Az␣átlag␣=␣%10.3f\n", atlag);
34      }
35      return 0;
36  }
```

$\binom{n}{k}$ [1]

nalattk.c [1–25]

```c
/* n alatt k értékének kiszámítása
 * 1997. Október 25.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

int main(int argc, char *argv[]) {
    int n, k, nak;                                          /* n alatt k */
    int i;                                                  /* ciklusváltozó */

    printf("Kérem n-t és k-t\n");
    printf("az n alatt k értékének kiszámításhoz\n");
    scanf("%d%d%*[^\n]", &n, &k); getchar();                /* beolvasás */

    if (n < k || k < 0) {                                   /* input adatok jók-e? */
        printf("%d alatt %d nem értelmezett!\n", n, k);     /* kiíratás */
    } else {
        nak = 1;                                            /* inicializálás */
        for (i = 1; i <= k; ++i) {                          /* ciklus */
            nak = nak * (n - i + 1) / i;
        }
        printf("%d alatt %d = %d\n", n, k, nak);            /* kiíratás */
    }
    return 0;
}
```

A `volatile` kulcsszó hiánya [1/1]

non-volatile.c [1–26]

```c
/* A volatile kulcsszó hatásának bemutatása
 * 2016. December 13.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void *szal(void *arg) {
    while (scanf("%d", (int*)arg) == 1);
    *((int*)arg) = 0;
    printf("Szál befejezése.\n");
    return NULL;
}

int main() {
    pthread_t kiiro_szal;
    int valtozo = 1;
    printf("A program akkor fejeződik be, ha egy 0 értékű input adat után egy\n"
           "egészként nem értelmezhető adatot adunk neki.\n");
    pthread_create(&kiiro_szal, NULL, szal, (void*)&valtozo);
    while (valtozo);
    printf("Várunk a szál befejezésére.\n");
    pthread_join(kiiro_szal, NULL);
    return 0;
}
```

Pascal háromszög ciklusos $\binom{n}{k}$ implementációval [1/2]

P3szog.c [1–25]

```c
/* n alatt k értékének kiszámítása egy nemrekurzív függvénnyel és
 *   az értékek elrendezése a Pascal háromszögben.
 * 1997. Október 31.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2013. Augusztus 29. Gergely Tamás, gertom@inf.u-szeged.hu
 * 2020. Július 30. Jász Judit, jasy@inf.u-szeged.hu
 */

#include <stdio.h>

#define SZAMSZ       5                      /* egy szám kiírási szélessége */
#define KEPSZ       80                          /* a képernyő szélessége */

int n_alatt_k(int n, int k) {
                /* n alatt k értékének kiszámítása nemrekurzív függvénnyel */
    int i, nak;

    if (n >= k && k >= 0) {                        /* input adatok jók-e? */
        nak = 1;                                       /* inicializálás */
        for (i = 1; i <= k; ++i)                               /* ciklus */
            nak = nak * (n - i + 1) / i;
    } else {
        nak = 0;
    }
    return nak;
}
```

Pascal háromszög ciklusos $\binom{n}{k}$ implementációval [2/2]

P3szog.c [27–45]

```c
int main() {
    int n;                                          /* a sorok száma */
    int i, j;                                       /* ciklusváltozók */

    printf("Kérem␣a␣Pascal␣háromszög␣sorainak␣számát\n");
    scanf("%d%*[^\n]", &n); getchar();              /* beolvasás */
    if (0 <= n && n <= KEPSZ / SZAMSZ - 2) {        /* kifér-e a képernyőre? */
        for (i = 0; i <= n; ++i) {
            printf("%*c", KEPSZ / 2 - (i + 1) * SZAMSZ / 2 - 1, '␣'); /* pozicionálás */
            for (j = 0; j <= i; ++j) {
                printf("%*d", SZAMSZ, n_alatt_k(i, j));   /* kiíratás */
            }
            putchar('\n');
        }
    } else {                                        /* hibaüzenet */
        printf("%d␣sor␣nem␣fér␣ki␣a␣␣képernyőre\n", n);
    }
    return 0;
}
```

Pascal háromszög rekurzív $\binom{n}{k}$ implementációval [1/2]

P3szogr.c [1–22]

```c
/* n alatt k értékének kiszámítása egy rekurzív függvénnyel és
 *   az értékek elrendezése a Pascal háromszögben.
 * 1997. Október 31. Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2015. Október 11. Gergely Tamás, gertom@inf.u-szeged.hu
 * 2020. Július 30. Jász Judit, jasy@inf.u-szeged.hu
 */

#include <stdio.h>

#define SZAMSZ       5                      /* egy szám kiírási szélessége */
#define KEPSZ       80                          /* a képernyő szélessége */

int n_alatt_k(int n, int k) {
                  /* n alatt k értékének kiszámítása rekurzív függvénnyel */
    if (n < k || k < 0) {                              /* input adatok jók-e? */
        return 0;
    }
    if (n <= 1 || n == k || k == 0) {                          /* alapesetek */
        return 1;
    }
    return n_alatt_k(n - 1, k - 1) + n_alatt_k(n - 1, k);    /* rek. hívás */
}
```

Pascal háromszög rekurzív $\binom{n}{k}$ implementációval [2/2]

P3szogr.c [24–42]

```
24 int main() {
25     int n;                                                  /* a sorok száma */
26     int i, j;                                              /* ciklusváltozók */
27
28     printf("Kérem␣a␣Pascal␣háromszög␣sorainak␣számát\n");
29     scanf("%d%*[^\n]", &n); getchar();                        /* beolvasás */
30     if (0 <= n && n <= KEPSZ / SZAMSZ - 2) {      /* kifér-e a képernyőre? */
31         for (i = 0; i <= n; ++i) {
32             printf("%*c", KEPSZ / 2 - (i + 1) * SZAMSZ / 2 - 1, '␣'); /* pozicionálás */
33             for (j = 0; j <= i; ++j) {
34                 printf("%*d", SZAMSZ, n_alatt_k(i, j));        /* kiíratás */
35             }
36             putchar('\n');
37         }
38     } else {                                                /* hibaüzenet */
39         printf("%d␣sor␣nem␣fér␣ki␣a␣␣képernyőre\n", n);
40     }
41     return 0;
42 }
```

Pénzváltás [1/1]

penzvalto.c [1–14]

```c
/* Pénzváltás rögzített árfolyamon.
 * 2012. Szeptember 5.  Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double forint, euro;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &forint);
    euro = forint / 347.981;                                    /* 2021-09-01 */
    printf("%lf HUF = %lf EUR\n", forint, euro);
    return 0;
}
```

Pénzváltás („szebb” implementáció) [1/1]

penzvalto-fgv.c [1–19]

```c
/* Pénzváltás rögzített árfolyamon, függvénnyel megvalósítva.
 * 2012. September 5.  Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define EURO 347.981                                    /* 2021-09-01 */

double valtas(double forint, double arfolyam) {
    return forint / arfolyam;
}

int main() {
    double osszeg;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &osszeg);
    printf("%lf HUF = %lf EUR\n", osszeg, valtas(osszeg, EURO));
    return 0;
}
```

Pénzváltás

penzvalto.c (v1.0) [1–16]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2021 Szeptember 1. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

int main() {
    double forint, euro;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &forint);
    euro = forint / 347.981;                                    /* 2021-09-01 */
    printf("%lf HUF = %lf EUR\n", forint, euro);
    return 0;
}
```

Pénzváltás

penzvalto.c (v2.0) [1–21]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2021 Szeptember 1. Gergely Tamás, gertom.inf.u-szeged.hu
 */

#include <stdio.h>

#define EUR 347.981                                    /* 2021-09-01 */

double valtas(double forint, double arfolyam) {
    return forint / arfolyam;
}

int main() {
    double osszeg;
    printf("Hány forintot váltsunk? ");
    scanf("%lf", &osszeg);
    printf("%lf HUF = %lf EUR\n", osszeg, valtas(osszeg, EUR));
    return 0;
}
```

Pénzváltás parancssorból

penzvalto.c (v3.0) [1–23]

```c
/* Pénzváltás rögzített árfolyamon.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2021 Szeptember 1. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define EUR 347.981                                    /* 2021-09-01 */

double valtas(double forint, double arfolyam) {
    return forint / arfolyam;
}

int main(int argc, char *argv[]) {
    for (int i = 1; i < argc; ++i) {
        double osszeg;
        osszeg = atoi(argv[i]);
        printf("%.2lf HUF = %.2lf EUR\n", osszeg, valtas(osszeg, EUR));
    }
    return 0;
}
```

π kiszámítása tetszőleges pontossággal [1/7]

pi.c [1–23]

```
1  /*
2  */
3  /*****************************************/
4  /* Compute pi to arbitrary precision     */
5  /* Author Roy Williams February 1994     */
6  /*   roy@ccsf.caltech.edu                */
7  /* Uses Machin's formula...              */
8  /* pi/4 = 4 arctan(1/5) - arctan(1/239)  */
9  /*****************************************/
10 /* compile with cc -O -o pi pi.c         */
11 /* run as "pi 1000" for 1000 digits      */
12 /*****************************************/
13 /* The last few digits may be wrong..... */
14
15 #include <stdio.h>
16 #include <memory.h>
17
18 #define BASE    10000
19 int nblock;
20 int *tot;
21 int *t1;
22 int *t2;
23 int *t3;
```

π kiszámítása tetszőleges pontossággal [2/7]

pi.c [25–53]

```
25 main(argc, argv)
26 int argc;
27 char **argv;
28 {
29         int ndigit = 60;
30         if(argc == 2)
31                 ndigit = atoi(argv[1]);
32         else {
33                 fprintf(stderr, "Usage: %s ndigit\n", argv[0]);
34                 fprintf(stderr, "(Assuming 60 digits)\n");
35         }
36         if(ndigit < 20) ndigit = 20;
37         nblock = ndigit/4;
38         tot = (int *)malloc(nblock*sizeof(int));
39         t1 = (int *)malloc(nblock*sizeof(int));
40         t2 = (int *)malloc(nblock*sizeof(int));
41         t3 = (int *)malloc(nblock*sizeof(int));
42         if(!tot || !t1 || !t2 || !t3){
43                 fprintf(stderr, "Not enough memory\n");
44                 exit(1);
45         }
46
47         arctan(tot, t1, t2, 5, 1);
48         mult(tot, 4);
49         arctan(t3, t1, t2, 239, 2);
50         sub(tot, t3);
51         mult(tot, 4);
52         print(tot);
53 }
```

π kiszámítása tetszőleges pontossággal [3/7]

pi.c [55–80]

```
55 arctan(result, w1, w2, denom, onestep)
56 int *result, *w1, *w2, denom, onestep;
57 {
58         int denom2 = denom*denom;
59         int k = 1;
60
61         set(result, 1);
62         div(result, denom);
63         copy(w1, result);
64
65         do{
66                 if(onestep)
67                         div(w1, denom2);
68                 else {
69                         div(w1, denom);
70                         div(w1, denom);
71                 }
72                 copy(w2, w1);
73                 div(w2, 2*k+1);
74                 if(k%2)
75                         sub(result, w2);
76                 else
77                         add(result, w2);
78                 k++;
79         } while(!zero(w2));
80 }
```

π kiszámítása tetszőleges pontossággal [4/7]

pi.c [82–98]

```
82 copy(result, from)
83 int *result, *from;
84 {
85         int i;
86         for(i=0; i<nblock; i++)
87                 result[i] = from[i];
88 }
89
90 zero(result)
91 int *result;
92 {
93         int i;
94         for(i=0; i<nblock; i++)
95                 if(result[i])
96                         return 0;
97         return 1;
98 }
```

π kiszámítása tetszőleges pontossággal [5/7]

pi.c [100–124]

```
100 add(result, increm)
101 int *result, *increm;
102 {
103         int i;
104         for(i=nblock-1; i>=0; i--){
105                 result[i] += increm[i];
106                 if(result[i] >= BASE){
107                         result[i] -= BASE;
108                         result[i-1]++;
109                 }
110         }
111 }
112
113 sub(result, decrem)
114 int *result, *decrem;
115 {
116         int i;
117         for(i=nblock-1; i>=0; i--){
118                 result[i] -= decrem[i];
119                 if(result[i] < 0){
120                         result[i] += BASE;
121                         result[i-1]--;
122                 }
123         }
124 }
```

π kiszámítása tetszőleges pontossággal [6/7]

pi.c [126–148]

```
126 mult(result, factor)
127 int *result, factor;
128 {
129         int i, carry = 0;
130         for(i=nblock-1; i>=0; i--){
131                 result[i] *= factor;
132                 result[i] += carry;
133                 carry = result[i]/BASE;
134                 result[i] %= BASE;
135         }
136 }
137
138 div(result, denom)
139 int *result, denom;
140 {
141         int i, carry = 0;
142
143         for(i=0; i<nblock; i++){
144                 result[i] += carry*BASE;
145                 carry = result[i] % denom;
146                 result[i] /= denom;
147         }
148 }
```

π kiszámítása tetszőleges pontossággal [7/7]

pi.c [150–173]

```
150 set(result, rhs)
151 int *result, rhs;
152 {
153         int i;
154         for(i=0; i<nblock; i++)
155                 result[i] = 0;
156         result[0] = rhs;
157 }
158
159 print(result)
160 int *result;
161 {
162         int i, k;
163         char s[10];
164         printf("%1d.\n", result[0]);
165         for(i=1; i<nblock; i++){
166                 sprintf(s, "%4d␣", result[i]);
167                 for(k=0; k<5; k++)
168                         if(s[k] == '␣') s[k] = '0';
169                 printf("%c%c%c%c", s[0], s[1], s[2], s[3]);
170                 if(i%15 == 0) printf("\n");
171         }
172         printf("\n");
173 }
```

Preprocesszálás [1]

preproc.c [1–20]

```c
/* Preprocesszálás hatásának bemutatása.
 * 2009. December 17.  Gergely Tamás, gertom@inf.u-szeged.hu
 */
#define N 30

#ifdef DEBUG
#define STRING "Debug"
#else
#define STRING "Release"
#endif

#line 200
int main() {
    int unix;
    char tomb[N] = STRING;
    for (unix = N - 1; unix && tomb[unix]; --unix) {
        tomb[unix] = 0;
    }
    return 0;
}
```

Prímszámok meghatározása [1/3]

prim.c [1–14]

```c
/* Határozzuk meg az adott N természetes számnál nem nagyobb
 *   prímszámokat.
 * 1997. December 6.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2006. Augusztus 15.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

#define K (8 * sizeof(kishalmaz_t))
#define M 262144LL
#define N (M * K)

typedef long long int kishalmaz_t;
typedef kishalmaz_t halmaz_t[M];
```

Prímszámok meghatározása [2/3]

prim.c [16–42]

```
16 int main() {
17     halmaz_t szita;
18     kishalmaz_t kicsi;                              /* a szita inicializálásához */
19     long long int p, s, lepes, i, j;
20
21     kicsi = 0;                                       /* a szita inicializálása */
22     for (i = 0; i <= ((K - 1) / 2); ++i) {
23         kicsi |= (1LL << (2 * i + 1));
24     }
25     for (i = 0; i < M; ++i) {
26         szita[i] = kicsi;
27     }
28     szita[0] &= ~2LL; /* 2LL == (1LL << 1) */
29     szita[0] |= 4LL;  /* 4LL == (1LL << 2) */
30
31     p = 3;                                           /* az első szitálandó prím */
32     while (p * p  < N) {                    /* P többszöröseinek kiszitálása */
33         lepes = 2 * p;                                    /* lépésköz = 2*p */
34         s = p * p;                                  /* s az első többszörös */
35         while (s < N) {
36             szita[s / K] &= ~(1LL << (s % K));
37             s += lepes;
38         }
39         do {                                       /* a következő prím keresése */
40             p += 2;
41         } while ((p < N) && !(szita[p / K] & (1LL << (p % K))));
42     }
```

Prímszámok meghatározása [3/3]

prim.c [44–57]

```
44      j = 0;                                  /* a prímszámok kiíratása képernyőre */
45      printf("A␣prímszámok␣%lld-ig:\n", N);
46      for (p = 2; p < N; ++p) {
47          if (szita[p / K] & (1LL << (p % K))) {
48              printf("%9lld", p);
49              if (++j == 8) {
50                  j = 0;
51                  putchar('\n');
52              }
53          }
54      }
55      putchar('\n');
56      return 0;
57  }
```

Prímszámok meghatározása a Halmaz modullal [1/2]

prim3.c [1–21]

```c
/* Határozzuk meg az adott N természetes számnál nem nagyobb
 *   prímszámokat.
 * 1997. December 6.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2006. Augusztus 16.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>
#include "halmaz.h"

#define N 16777216LL

int main() {
    halmaz_t szita;
    halmazelem_t p, s, i;
    long long int lepes, j;

    szita = letesit(N);
    bovit(szita, 2);                          /* a szita inicializálása */
    for (i = 3; i <= N; i += 2) {
        bovit(szita, i);
    }
```

Prímszámok meghatározása a Halmaz modullal [2/2]

prim3.c [23–50]

```
23      p = 3;                                          /* az első szitálandó prím */
24      while (p * p  <= N) {                       /* P többszöröseinek kiszitálása */
25          lepes = 2 * p;                                      /* lépésköz = 2*p */
26          s = p * p;                                    /* s az első többszörös */
27          while (s <= N) {
28              torol(szita, s);
29              s += lepes;
30          }
31          do {                                        /* a következő prím keresése */
32              p += 2;
33          } while ((p < N) && (! eleme(szita, p)));
34      }
35
36      j = 0;                                  /* a prímszámok kiíratása képernyőre */
37      printf("A␣prímszámok␣%lld-ig:\n", N);
38      for (p = 2; p < N; ++p) {
39          if (eleme(szita, p)) {
40              printf("%9" HALMAZELEM_PRINT_FORMAT, p);
41              if (++j == 8) {
42                  putchar('\n');
43                  j = 0;
44              }
45          }
46      }
47      putchar('\n');
48      megszuntet(szita);
49      return 0;
50  }
```

Prímszámok meghatározása [1/2]

prim-nem-hatekony.c [1–18]

```c
/* Határozzuk meg az adott N természetes számnál nem nagyobb
 *   prímszámokat.
 * 2014. Január 24.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

#define N 16777216LL

int prim_e(long long int n) {
    long long int d;
    for (d = 2; d * d <= n; ++d) {
        if (n % d == 0) {
            return 0;
        }
    }
    return 1 < n;
}
```

Prímszámok meghatározása [2/2]

prim-nem-hatekony.c [20–34]

```
20 int main() {
21     long long int p, j = 1;
22     printf("A␣prímszámok␣%lld-ig:\n%9lld", N, 2LL);
23     for (p = 3; p < N; p += 2) {
24         if (prim_e(p)) {
25             printf("%9lld", p);
26             if (++j == 8) {
27                 j = 0;
28                 putchar('\n');
29             }
30         }
31     }
32     putchar('\n');
33     return 0;
34 }
```

Riadólánc [1/2]

riadolanc.c [1–18]

```c
/* Adott n számú embert tartalmazó közösség. Eldöntendő,
 *   hogy riadóláncot alkotnak-e?
 * 2006. augusztus 9. Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>

#define N    8                                  /* a sorozat elemeinek száma */
#define ELSO 0                          /* az első vizsgálandó elem sorszáma */

int main () {
    int lanc[N];                                 /* a sorozatot tároló tömb */
    int e, i;                                          /* munkaváltozók */
    int kov;                                   /* a következő vizsgálandó */
    int szam;                                             /* számláló */

    printf("Kérem␣a␣%d␣eleműű␣sorozatot␣,␣amelyről␣", N);
    printf("eldöntöm,␣hogy␣riadólánc-e!\n");
```

Riadólánc [2/2]

riadolanc.c [20–46]

```
20      for (i = 0; i < N; ++i) {                                   /* beolvasás */
21          printf("%d. kit értesít? ", i);
22          scanf("%d%*[^\n]", &e); getchar();
23
24          while ((e < 0) || (N <= e)) {
25              printf("Hibás adat!\nKérem újra: ");
26              scanf("%d%*[^\n]", &e); getchar();
27          }
28          lanc[i] = e;
29      }
30
31      kov  = ELSO;                                              /* inicializálás */
32      szam = 0;
33      do {
34          i       = kov;
35          kov     = lanc[i];                                      /* továbblépés */
36          lanc[i] = N;                                /* jártam már itt bejegyzése */
37          ++szam;
38      } while (lanc[kov] != N);                                   /* jártam már itt? */
39
40      if ((szam == N) && (kov == ELSO)) {
41          printf("A számsorozat riadólánc.\n");
42      } else {
43          printf("A számsorozat nem riadólánc.\n");
44      }
45      return 0;
46  }
```

Ciklikus függvény [1/2]

riadolanc-ciklikus.c [1–21]

```c
/* Adott n számú embert tartalmazó közösség. Eldöntendő,
 *   hogy riadóláncot alkotnak-e?
 *   Ehhez felhasználunk egy függvényt, amely eldönti, hogy
 *   a kapott sorozat ciklikus permutáció-e?
 * 2005. október 13. Gergely Tamás, gertom@inf.u-szeged.hu
 * 2014. október 9. Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <stdio.h>
#include <stdbool.h>

#define N    8                                  /* a sorozat elemeinek száma */
#define ELSO 0                          /* az első vizsgálandó elem sorszáma */

bool ciklikus(int n, int perm[]) {                 /* A paraméter egy tömb */
    int kov = ELSO, szam = 0;                     /* következő és számláló */
    do {
        kov = perm[kov]; szam++;                                /* továbblépés */
    } while ((kov != ELSO) && (szam != n));
    return (kov == ELSO) && (szam == n);
}
```

Ciklikus függvény [2/2]

riadolanc-ciklikus.c [23–47]

```c
int main () {
    int lanc[N];                                    /* a sorozatot tároló tömb */
    int e, i;                                                /* munkaváltozók */

    printf("Kérem␣a␣%d␣elemű␣sorozatot,␣amelyről␣", N);
    printf("eldöntöm,␣hogy␣riadólánc-e!\n");

    for (i = 0; i < N; ++i) {                                    /* beolvasás */
        printf("%d.␣kit␣értesít␣?␣", i);
        scanf("%d%*[^\n]", &e); getchar();

        while ((e < 0) || (N <= e)) {
            printf("Hibás␣adat!\nKérem␣újra:␣");
            scanf("%d%*[^\n]", &e); getchar();
        }
        lanc[i] = e;
    }

    if (ciklikus(N, lanc)) {
        printf("A␣számsorozat␣riadólánc.\n");
    } else {
        printf("A␣számsorozat␣nem␣riadólánc.\n");
    }
    return 0;
}
```

Röppálya határértékeinek számítása [1/1]

roppalya.c [1–26]

```c
/* Röppálya közelítő számítása.
 * 2012. Szeptember 5.  Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 9.80665

int main() {
    double v0, vx, vy, alfa, t, sx, sy;
    printf("Kezdősebesség (m/s)? "); scanf("%lf", &v0);
    printf("Szög (fok)? "); scanf("%lf", &alfa);
    if (0.0 <= alfa && alfa <= 90.0 && 0.0 <= v0) {
        vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
        t  = vy / G;
        sy = G / 2.0 * t * t;
        printf("A röppálya legnagyobb magassága: %lf\n", sy);
        vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
        sx = vx * 2.0 * t;
        printf("A lövedék távolsága földetéréskor: %lf\n", sx);
    } else {
        printf("Hibás adatok!\n");
    }
    return 0;
}
```

Röppálya határértékeinek számítása

roppalya.c (v1.0) [1–28]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 9.80665

int main() {
    double v0, vx, vy, alfa, t, sx, sy;
    printf("Kezdősebesség (m/s)? "); scanf("%lf", &v0);
    printf("Szög (fok)? "); scanf("%lf", &alfa);
    if (0.0 <= alfa && alfa <= 90.0 && 0.0 <= v0) {
        vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
        t  = vy / G;
        sy = G / 2.0 * t * t;
        printf("A röppálya legnagyobb magassága: %lf m\n", sy);
        vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
        sx = vx * 2.0 * t;
        printf("A lövedék távolsága földetéréskor: %lf m\n", sx);
    } else {
        printf("Hibás adatok!\n");
    }
    return 0;
}
```

Röppálya görbe szimuláció [1/2]

roppalya-struct.c [1–13]

```c
/* Röppálya közelítő számítása.
 * 2012. Szeptember 5.  Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 9.80665

typedef struct {
    double x;
    double y;
} vektor_t;
```

Röppálya görbe szimuláció [2/2]

roppalya-struct.c [15–32]

```
15 int main() {
16     double v0, alfa, dt, t;
17     vektor_t v, s;
18     s.x = s.y = t = 0.0;
19     printf("Kezdősebesség␣(m/s)?␣"); scanf("%lf", &v0);
20     printf("Szög␣(fok)?␣"); scanf("%lf", &alfa);
21     printf("Delta␣t␣(s)?␣"); scanf("%lf", &dt);
22     v.x = v0 * cos(alfa / 90.0 * M_PI / 2.0);
23     v.y = v0 * sin(alfa / 90.0 * M_PI / 2.0);
24     while (s.y >= 0.0) {
25         printf("%lf␣sec:␣poz(%lf,␣%lf);␣seb[%lf,␣%lf]\n", t, s.x, s.y, v.x, v.y);
26         s.x += v.x * dt;
27         s.y += v.y * dt - G / 2.0 * dt * dt;
28         v.y -= G * dt;
29         t   += dt;
30     }
31     return 0;
32 }
```

Röppálya szimuláció

roppalya-szim.c (v1.0) [1–15]

```c
/* Röppálya közelítő számítása.
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

#define G 9.80665

typedef struct {
    double x;
    double y;
} vektor_t;
```

Röppálya szimuláció

roppalya-szim.c (v1.0) [17–34]

```
17 int main() {
18     double v0, alfa, dt, t;
19     vektor_t v, s;
20     s.x = s.y = t = 0.0;
21     printf("Kezdősebesség␣(m/s)?␣"); scanf("%lf", &v0);
22     printf("Szög␣(fok)?␣"); scanf("%lf", &alfa);
23     printf("Delta␣t␣(s)?␣"); scanf("%lf", &dt);
24     v.x = v0 * cos(alfa / 90.0 * M_PI / 2.0);
25     v.y = v0 * sin(alfa / 90.0 * M_PI / 2.0);
26     while (s.y >= 0.0) {
27         printf("%lf␣sec:␣poz(%lf,␣%lf);␣seb[%lf,␣%lf]\n", t, s.x, s.y, v.x, v.y);
28         s.x += v.x * dt;
29         s.y += v.y * dt - G / 2.0 * dt * dt;
30         v.y -= G * dt;
31         t   += dt;
32     }
33     return 0;
34 }
```

Síkidomok 1. verzió [1/4]

sikidomok.c [1–24]

```c
/* Síkidomok területének és kerületének kiszámítása.
 * 2018. Szeptember 19. Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

#define PUFFERMERET 128

enum sikidom_tipus_t {
    kor, haromszog, teglalap
};

struct kor_tulajdonsagok_t {
    double r;
};

struct haromszog_tulajdonsagok_t {
    double a, b, c;
};

struct teglalap_tulajdonsagok_t {
    double a, b;
};
```

Síkidomok 1. verzió [2/4]

sikidomok.c [26–54]

```c
struct sikidom {
    enum sikidom_tipus_t tipus;
    union {
        struct kor_tulajdonsagok_t kor;
        struct haromszog_tulajdonsagok_t haromszog;
        struct teglalap_tulajdonsagok_t teglalap;
    };
};

double kor_kerulet(struct kor_tulajdonsagok_t k) {
    return 2.0 * M_PI * k.r;
}

double kor_terulet(struct kor_tulajdonsagok_t k) {
    return M_PI * k.r * k.r;
}

double haromszog_kerulet(struct haromszog_tulajdonsagok_t h) {
    return h.a + h.b + h.c;
}

double haromszog_terulet(struct haromszog_tulajdonsagok_t h) {
    double s = (h.a + h.b + h.c) / 2.0;
    return sqrt(s * (s-h.a) * (s-h.b) * (s-h.c));
}

double teglalap_kerulet(struct teglalap_tulajdonsagok_t t) {
    return 2.0 * (t.a + t.b);
}
```

Síkidomok 1. verzió [3/4]

sikidomok.c [56–76]

```
56 double teglalap_terulet(struct teglalap_tulajdonsagok_t t) {
57     return t.a * t.b;
58 }
59
60 double kerulet(struct sikidom s) {
61     switch (s.tipus) {
62     case kor:       return kor_kerulet(s.kor);
63     case haromszog: return haromszog_kerulet(s.haromszog);
64     case teglalap:  return teglalap_kerulet(s.teglalap);
65     default:        return NAN;
66     }
67 }
68
69 double terulet(struct sikidom s) {
70     switch (s.tipus) {
71     case kor:       return kor_terulet(s.kor);
72     case haromszog: return haromszog_terulet(s.haromszog);
73     case teglalap:  return teglalap_terulet(s.teglalap);
74     default:        return NAN;
75     }
76 }
```

Síkidomok 1. verzió [4/4]

sikidomok.c [78–103]

```
78 char puffer[PUFFERMERET];
79
80 int main() {
81     double a,b,c;
82     struct sikidom s;
83     while (fgets(puffer, PUFFERMERET, stdin)) {
84         if (sscanf(puffer, "kor(%lf)", &a) == 1) {
85             s.tipus = kor;
86             s.kor.r = a;
87         } else if (sscanf(puffer, "teglalap(%lf,%lf)", &a, &b) == 2) {
88             s.tipus = teglalap;
89             s.teglalap.a = a;
90             s.teglalap.b = b;
91         } else if (sscanf(puffer, "haromszog(%lf,%lf,%lf)", &a, &b, &c) == 3) {
92             s.tipus = haromszog;
93             s.haromszog.a = a;
94             s.haromszog.b = b;
95             s.haromszog.c = c;
96         } else {
97             printf("Ismeretlen␣formatumu␣sor!\n");
98             continue;
99         }
100        printf("T=␣%lf\nK=␣%lf\n", terulet(s), kerulet(s));
101    }
102    return 0;
103 }
```

Síkidomok 2. verzió [1/5]

sikidomok-fgv-ptr.c [1–12]

```c
/* Síkidomok területének és kerületének kiszámítása.
 * 2018. Szeptember 19. Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

#define PUFFERMERET 128

enum sikidom_tipus_t {
    kor, haromszog, teglalap
};
```

Síkidomok 2. verzió [2/5]

sikidomok-fgv-ptr.c [14–35]

```
14 struct kor_tulajdonsagok_t {
15     double r;
16 };
17
18 struct haromszog_tulajdonsagok_t {
19     double a, b, c;
20 };
21
22 struct teglalap_tulajdonsagok_t {
23     double a, b;
24 };
25
26 struct sikidom {
27     enum sikidom_tipus_t tipus;
28     union {
29         struct kor_tulajdonsagok_t kor;
30         struct haromszog_tulajdonsagok_t haromszog;
31         struct teglalap_tulajdonsagok_t teglalap;
32     };
33     double (*kerulet)(struct sikidom);
34     double (*terulet)(struct sikidom);
35 };
```

Síkidomok 2. verzió [3/5]

sikidomok-fgv-ptr.c [37–62]

```
37 double kor_kerulet(struct sikidom si) {
38     return (si.tipus == kor) ? 2.0 * M_PI * si.kor.r : NAN;
39 }
40
41 double kor_terulet(struct sikidom si) {
42     return (si.tipus == kor) ? M_PI * si.kor.r * si.kor.r : NAN;
43 }
44
45 double haromszog_kerulet(struct sikidom si) {
46     return (si.tipus == haromszog) ?
47             si.haromszog.a + si.haromszog.b + si.haromszog.c : NAN;
48 }
49
50 double haromszog_terulet(struct sikidom si) {
51     double s = (si.haromszog.a + si.haromszog.b + si.haromszog.c) / 2.0;
52     return (si.tipus == haromszog) ?
53             sqrt(s * (s-si.haromszog.a) * (s-si.haromszog.b) * (s-si.haromszog.c)) : NAN;
54 }
55
56 double teglalap_kerulet(struct sikidom si) {
57     return (si.tipus == teglalap) ? 2.0 * (si.teglalap.a + si.teglalap.b) : NAN;
58 }
59
60 double teglalap_terulet(struct sikidom si) {
61     return (si.tipus == teglalap) ? si.teglalap.a * si.teglalap.b : NAN;
62 }
```

Síkidomok 2. verzió [4/5]

sikidomok-fgv-ptr.c [64–92]

```
64 struct sikidom uj_kor(double r) {
65     struct sikidom s;
66     s.tipus   = kor;
67     s.kor.r   = r;
68     s.kerulet = kor_kerulet;
69     s.terulet = kor_terulet;
70     return s;
71 }
72
73 struct sikidom uj_teglalap(double a, double b) {
74     struct sikidom s;
75     s.tipus      = teglalap;
76     s.teglalap.a = a;
77     s.teglalap.b = b;
78     s.kerulet    = teglalap_kerulet;
79     s.terulet    = teglalap_terulet;
80     return s;
81 }
82
83 struct sikidom uj_haromszog(double a, double b, double c) {
84     struct sikidom s;
85     s.tipus       = haromszog;
86     s.haromszog.a = a;
87     s.haromszog.b = b;
88     s.haromszog.c = c;
89     s.kerulet     = haromszog_kerulet;
90     s.terulet     = haromszog_terulet;
91     return s;
92 }
```

Síkidomok 2. verzió [5/5]

sikidomok-fgv-ptr.c [94–113]

```
94 char puffer[PUFFERMERET];
95
96 int main() {
97     double a,b,c;
98     struct sikidom s;
99     while (fgets(puffer, PUFFERMERET, stdin)) {
100        if (sscanf(puffer, "kor(%lf)", &a) == 1) {
101            s = uj_kor(a);
102        } else if (sscanf(puffer, "teglalap(%lf,%lf)", &a, &b) == 2) {
103            s = uj_teglalap(a, b);
104        } else if (sscanf(puffer, "haromszog(%lf,%lf,%lf)", &a, &b, &c) == 3) {
105            s = uj_haromszog(a, b, c);
106        } else {
107            printf("Ismeretlen formatumu sor!\n");
108            continue;
109        }
110        printf("T= %lf\nK= %lf\n", s.terulet(s), s.kerulet(s));
111    }
112    return 0;
113 }
```

A `static` kulcsszó hatása [1/1]

static.c [1–23]

```c
/* A static változót mutatjuk be.
 * 1997. November 7.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

#include <stdio.h>

void stat();

int main() {
    int i;                                          /* ciklusváltozó */
    for (i = 0; i < 5; ++i) {
        stat();
    }
    return 0;
}

void stat() {
    int ideiglenes = 1;                 /* minden hívásnál inicializálódik */
    static int allando = 1;    /* a program elején egyszer inicializálódik */
    printf("ideiglenes = %d allando = %d\n", ideiglenes, allando);
    ++ideiglenes;
    ++allando;
}
```

Sztring hosszának meghatározása [1/4]

strlen.c [1–21]

```c
/* A karaktersorozat hosszának meghatározására függvényműveletet írtunk:
 * 2009. December 17.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

int strlen1(char s[]) {                          /* s hosszának kiszámítása */
    int i = 0;
    while (s[i] != '\0') {
        ++i;
    }
    return i;
}

/*
 * Egy másik lehetséges megvalósítás ez volt:
 */

int strlen2(char s[]) {                          /* s hosszának kiszámítása */
    int i;
    for (i = 0; s[i] != '\0'; ++i);
    return i;
}
```

Sztring hosszának meghatározása [2/4]

strlen.c [23–39]

```
23 /*
24  * A karaktersorozatra pointer is mutathat:
25  */
26
27 int strlen3(char *s) {                          /* s hosszának kiszámítása */
28     int n;
29     for (n = 0; *s != '\0'; ++s) {
30         ++n;
31     }
32     return n;
33 }
34
35 int strlen4(char *s) {                          /* s hosszának kiszámítása */
36     int n;
37     for (n = 0; *s != '\0'; ++s, ++n);
38     return n;
39 }
```

Sztring hosszának meghatározása [3/4]

strlen.c [41–65]

```
41 /*
42  * További függvénydeklarációk a string hosszának meghatározására
43  */
44
45 int strlen5(char *s) {                          /* s hosszának kiszámítása */
46     char *p = s;
47     while (*p != '\0') {
48         ++p;
49     }
50     return p - s;
51 }
52
53 int strlen6(char *s) {                          /* s hosszának kiszámítása */
54     char *p = s;
55     while (*p) {
56         ++p;
57     }
58     return p - s;
59 }
60
61 int strlen7(char *s) {                          /* s hosszának kiszámítása */
62     char *p = s;
63     while (*p) ++p;
64     return p - s;
65 }
```

Sztring hosszának meghatározása [4/4]

strlen.c [67–77]

```
67 int strlen8(char *s) {                          /* s hosszának kiszámítása */
68     char *p;
69     for (p = s; *p != '\0'; ++p);
70     return p - s;
71 }
72
73 int strlen9(char *s) {                          /* s hosszának kiszámítása */
74     char *p;
75     for (p = s; *p; ++p);
76     return p - s;
77 }
```

Tömb és pointer különbsége [1/1]

strptr.c [1–24]

```c
/* A char* és char[] típusok különbsége inicializálás esetén.
 *   A program "pointer" paraméterrel történő indítása futási hibát okoz.
 * 2006. Augusztus 17. Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

int main(int argc, char *argv[]) {
    char t[] = "Tömb vagy pointer";
    char  *p = "Tömb vagy pointer";
    if (argc == 2) {
        if (!strcmp(argv[1], "tomb")) {
            strcpy(t, "Hello!");
        } else if (!strcmp(argv[1], "pointer")) {
            strcpy(p, "Hello!");
        }
        printf("p: \"%s\"\n", p);
        printf("t: \"%s\"\n", t);
    } else {
        printf("Használat:\n\t%s tomb\n\t%s pointer\n", argv[0], argv[0]);
    }
    return 0;
}
```

Szinusz(x) kiszámítása [1/2]

szinusz.c [1–22]

```c
/* sin(x) közelítő értékének kiszámítása a beépített sin(x)
 *    függvény alkalmazása nélkül.
 *    x értékét transzformáljuk a (-Pi,Pi) intervallumba.
 * 1997. Október 25.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

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

#define EPSZ    1e-10                          /* a közelítés pontossága */

int main() {
    double osszeg;                           /* a végtelen sor kezdőösszege */
    double tag;                             /* a végtelen sor aktuális tagja */
    double x;                                                 /* argumentum */
    double x_orig;                        /* az argumentum értékének megőrzése*/
    double xx;                                                  /* sqr(x) */
    int j;                                      /* a nevező kiszámításához */

    printf("Kérem␣sin(x)-hez␣az␣argumentumot\n");
    scanf("%lg%*[^\n]", &x); getchar();              /* sor végéig olvasunk */
    x_orig = x;
```

Szinusz(x) kiszámítása [2/2]

szinusz.c [24–44]

```
24      while (x < -M_PI) {                                     /* transzformálás */
25          x += 2 * M_PI;
26      }
27      while (M_PI < x) {
28          x -= 2 * M_PI;
29      }
30
31      osszeg = 0.0;
32      j      = 2;                                              /* inicializálás */
33      tag    = x;
34      xx     = x * x;
35
36      do {                                                    /* ciklus kezdete */
37          osszeg += tag;
38          tag = -(tag * xx / j / (j + 1));                    /* következő tag */
39          j += 2;
40      } while (fabs(tag) >= EPSZ);                /* végfeltétel, ciklus vége */
41
42      printf("sin(%8.5f)= %13.10f\n", x_orig, osszeg);
43      return 0;
44  }
```

Szinusz(x) kiszámítása, feltételes nyomkövetéssel [1/2]

szinusz-dbg.c [1–27]

```c
/* sin(x) közelítő értékének kiszámítása a beépített sin(x)
 *   függvény alkalmazása nélkül.
 * 1997. Október 25.  Dévényi Károly, devenyi@inf.u-szeged.hu
 * 2014. Május 19. Gergely Tamás, gertom@inf.u-szeged.hu
 */

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

#define EPSZ    1e-10                          /* a közelítés pontossága */

double szinusz(double x) {
    double osszeg = 0.0;                     /* a végtelen sor kezdőösszege */
    double tag    = x;                     /* a végtelen sor aktuális tagja */
    double xx     = x * x;                                    /* sqr(x) */
    int    j      = 2;                        /* a nevező kiszámításához */

    do {                                                  /* ciklus kezdete */
        osszeg += tag;
#if DEBUG > 0
        fprintf(stderr, "[LOG] Tag= %30.10lf Osszeg= %30.10lf\n", tag, osszeg);
#endif
        tag = -(tag * xx / j / (j + 1));                  /* következő tag */
        j += 2;
    } while (fabs(tag) >= EPSZ);              /* végfeltétel, ciklus vége */
    return osszeg;
}
```

Szinusz(x) kiszámítása, feltételes nyomkövetéssel [2/2]

szinusz-dbg.c [29–49]

```
29 int main() {
30     double x;                                                /* argumentum */
31
32     printf("Kérem␣sin(x)-hez␣az␣argumentumot\n");
33     scanf("%lg%*[^\n]", &x); getchar();                  /* egész sort olvasunk */
34 #ifdef DEBUG
35     printf("sin(%8.5f)=␣%13.10f\n", x, szinusz(x));
36 #endif
37
38     double x_orig = x;                                    /* eredeti argumentum */
39
40     while (x < -M_PI) {                                       /* transzformálás */
41         x += 2 * M_PI;
42     }
43     while (M_PI < x) {
44         x -= 2 * M_PI;
45     }
46     printf("sin(%8.5f)=␣%13.10f\n", x_orig, szinusz(x));
47
48     return 0;
49 }
```

Szinusz(x) kiszámítása függvényben megvalósítva [1/2]

szinusz-fgv.c [1–23]

```c
/* sin(x) közelítő értékének kiszámítása a beépített sin(x)
 *   függvény alkalmazása nélkül.
 *   X értékét transzformáljuk a (-Pi,Pi) intervallumba.
 * 1997. Október 25.  Dévényi Károly, devenyi@inf.u-szeged.hu
 */

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

#define EPSZ    1e-10                          /* a közelítés pontossága */

double szinusz(double x) {
    double osszeg = 0.0;                     /* a végtelen sor kezdőösszege */
    double tag;                             /* a végtelen sor aktuális tagja */
    double xx;                                                  /* sqr(x) */
    int    j      = 2;                       /* a nevező kiszámításához */

    while (x < -M_PI) {                                  /* transzformálás */
        x += 2 * M_PI;
    }
    while (M_PI < x) {
        x -= 2 * M_PI;
    }
```

Szinusz(x) kiszámítása függvényben megvalósítva [2/2]

szinusz-fgv.c [25–44]

```
25      tag = x;                                                /* inicializálás */
26      xx = x * x;
27
28      do {                                                   /* ciklus kezdete */
29          osszeg += tag;
30          tag = -(tag * xx / j / (j + 1));                    /* következő tag */
31          j += 2;
32      } while (fabs(tag) >= EPSZ);                /* végfeltétel, ciklus vége */
33
34      return osszeg;
35  }
36
37  int main() {
38      double x;                                                 /* argumentum */
39
40      printf("Kérem␣sin(x)-hez␣az␣argumentumot\n");
41      scanf("%lg%*[^\n]", &x); getchar();              /* egész sort olvasunk */
42      printf("sin(%8.5f)=␣%13.10f\n", x, szinusz(x));
43      return 0;
44  }
```

A tér legtávolabbi pontjai [1/3]

ter.c [1–17]

```c
/* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbit!
 * 2012. Szeptember 5.  Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

typedef struct {              /* A tér egy háromdimenziós pontjának típusa */
    double x, y, z;
} pont_t;

double tav(pont_t a, pont_t b) {
    return sqrt((a.x - b.x) * (a.x - b.x) +
                (a.y - b.y) * (a.y - b.y) +
                (a.z - b.z) * (a.z - b.z));
}
```

A tér legtávolabbi pontjai [2/3]

ter.c [19–32]

```
19 void legtavolabb(int n, pont_t p[], pont_t * egyik, pont_t * masik) {
20     double max = -1.0;
21     int i;
22     for (i = 0; i < n - 1; ++i) {
23         int j;
24         for (j = i + 1; j < n; ++j) {
25             if (tav(p[i], p[j]) > max) {
26                 max = tav(p[i], p[j]);
27                 *egyik = p[i];
28                 *masik = p[j];
29             }
30         }
31     }
32 }
```

A tér legtávolabbi pontjai [3/3]

ter.c [34–48]

```
34 int main() {
35     int darab, i;
36     pont_t *tomb, p1, p2;
37     scanf("%d", &darab);
38     tomb = malloc(darab * sizeof(pont_t));
39     for (i = 0; i < darab; ++i) {
40         scanf("%lf␣%lf␣%lf", &tomb[i].x, &tomb[i].y, &tomb[i].z);
41     }
42     legtavolabb(darab, tomb, &p1, &p2);
43     printf("A␣legtávolabbi␣pontok:\n");
44     printf("␣␣␣␣␣␣(%lf,␣%lf,␣%lf)␣és\n", p1.x, p1.y, p1.z);
45     printf("␣␣␣␣␣␣(%lf,␣%lf,␣%lf)\n",     p2.x, p2.y, p2.z);
46     free(tomb);
47     return 0;
48 }
```

A tér legtávolabbi pontjai [main függvény változat]

ter.c [35–49] helyett

```
35 int main() {
36     int darab, i;
37     pont *tomb, *t, P1, P2;
38     scanf("%d", &darab);
39     tomb = malloc(darab * sizeof(pont));
40     for (i = 0, t = tomb; i < darab; ++i, ++t) {
41         scanf("%lf␣%lf␣%lf", &t->x, &t->y, &t->z);
42     }
43     legtavolabb(darab, tomb, &P1, &P2);
44     printf("A␣legtávolabbi␣pontok:\n");
45     printf("␣␣␣␣␣␣(%lf,␣%lf,␣%lf)␣és\n", P1.x, P1.y, P1.z);
46     printf("␣␣␣␣␣␣(%lf,␣%lf,␣%lf)\n",     P2.x, P2.y, P2.z);
47     free(tomb);
48     return 0;
49 }
```

Tér legtávolabbi pontjai

ter.c (v1.0) [1–29]

```c
/* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbit!
 *
 * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
 * 2018. Július 26. Gergely Tamás, gertom.inf.u-szeged.hu
 */

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

typedef struct {              /* A tér egy háromdimenziós pontjának típusa */
    double x, y, z;
} pont_t;

pont_t read_pont() {
    pont_t p = {0.0, 0.0, 0.0};
    scanf("%lf %lf %lf", &p.x, &p.y, &p.z);
    return p;
}

void write_pont(pont_t p) {
    printf("(%lf, %lf, %lf)", p.x, p.y, p.z);
}

double tav(pont_t p, pont_t q) {
    return sqrt((p.x - q.x) * (p.x - q.x) +
                (p.y - q.y) * (p.y - q.y) +
                (p.z - q.z) * (p.z - q.z));
}
```

Tér legtávolabbi pontjai

ter.c (v1.0) [31–58]

```
31 void legtavolabb(int n, pont_t p[], pont_t *egyik, pont_t *masik) {
32     double max = -1.0;
33     for (int i = 0; i < n - 1; ++i) {
34         for (int j = i + 1; j < n; ++j) {
35             if (tav(p[i], p[j]) > max) {
36                 max = tav(p[i], p[j]);
37                 *egyik = p[i];
38                 *masik = p[j];
39             }
40         }
41     }
42 }
43
44 int main() {
45     int darab, i;
46     pont_t *tomb, p1, p2;
47     scanf("%d", &darab);
48     tomb = malloc(darab * sizeof(pont_t));
49     for (i = 0; i < darab; ++i) {
50         tomb[i] = read_pont();
51     }
52     legtavolabb(darab, tomb, &p1, &p2);
53     printf("A␣legtávolabbi␣pontok:\n");
54     printf("␣␣␣␣␣␣"); write_pont(p1); printf("␣és\n");
55     printf("␣␣␣␣␣␣"); write_pont(p2); putchar('\n');
56     free(tomb);
57     return 0;
58 }
```

A `volatile` kulcsszó hatása [1/1]

volatile.c [1–26]

```c
/* A volatile kulcsszó hatásának bemutatása
 * 2016. December 13.  Gergely Tamás, gertom@inf.u-szeged.hu
 */

#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void *szal(void *arg) {
    while (scanf("%d", (int*)arg) == 1);
    *((int*)arg) = 0;
    printf("Szál befejezése.\n");
    return NULL;
}

int main() {
    pthread_t kiiro_szal;
    volatile int valtozo = 1;
    printf("A program akkor fejeződik be, ha egy 0 értékű input adat után egy\n"
           "egészként nem értelmezhető adatot adunk neki.\n");
    pthread_create(&kiiro_szal, NULL, szal, (void*)&valtozo);
    while (valtozo);
    printf("Várunk a szál befejezésére.\n");
    pthread_join(kiiro_szal, NULL);
    return 0;
}
```

Hol vagyunk? [1/1]

where.c [1–12]

```
1            main(l
2        ,a,n,d)char**a;{
3   for(d=atoi(a[1])/10*80-
4  atoi(a[2])/5-596;n="@NKA\
5CLCCGZAAQBEAADAFaISADJABBA^\
6SNLGAQABDAXIMBAACTBATAHDBAN\
7ZcEMMCCCCAAhEIJFAEAAABAfHJE\
8TBdFLDAANEfDNBPHdBcBBBEA_AL\
9␣H␣E␣L␣L␣O,␣␣␣␣W␣O␣R␣L␣D!␣"
10    [l++-3];)for(;n--->64;)
11       putchar(!d+++33^
12            l&1);}
```