

Programozás Alapjai

Előadásprogramok

Dr. Gergely Tamás
Dr. Jász Judit

Szegedi Tudományegyetem
Informatikai Intézet
Szoftverfejlesztés Tanszék

2023

(v0914)



ABC [1/1]

abc.c [1-17]

```
1 /* Az ábécé kis- és nagybetűinek kiírása összefésülve.
2  * A for ciklusban a , műveletet alkalmazzuk.
3  * 1997. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
4  */
5
6 #include <stdio.h>
7
8 int main() {
9     char cha;                /* az ábécé kisbetűinek */
10    char chA;                /* az ábécé nagybetűinek */
11
12    for (cha = 'a', chA = 'A'; cha <= 'z'; ++cha, ++chA) {
13        printf("%c%c", cha, chA);
14    }
15    printf("\n");
16    return 0;
17 }
```



Angol ABC betűinek összefésült kiírása

abc.c (v1.0) [1–15]

```
1 /* Írassuk ki az angol ábécé kis- és nagybetűit összefésülve.
2  *
3  * 1997. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2018. Július 24. Gergely Tamás, gertom@inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 int main() {
10     for (char cha = 'a', chA = 'A'; cha <= 'z'; ++cha, ++chA) {
11         printf("%c%c", cha, chA);
12     }
13     printf("\n");
14     return 0;
15 }
```



Parancssori argumentumok kezelése [1/1]

arg.c [1-21]

```
1 /* Kiírjuk a parancssorban lévő argumentumokat.
2  * 1998. Április 16. Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int main(int argc, char **argv) {
8
9     /* vagy így is lehet az argv-t deklarálni
10
11     int main(int argc, char *argv[]) {
12
13     */
14         int i;
15
16         printf("argc=%d\n", argc);
17         for (i = 0; i < argc; ++i) {
18             printf("argv[%d]:%s\n", i, argv[i]);
19         }
20         return 0;
21 }
```

Egy útvonal leírásból a fájlnev kinyerése [1/1]

basename.c [1-26]

```
1 /* Egy útvonalból a fájlnev kinyerése.
2  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 #define N 1024
8
9 int main() {
10     char path[N], base[N];
11     int i, lastsep = -1;
12     fgets(path, N, stdin);
13     for (i = 0; path[i] != 0; ++i) {
14         if (path[i] == '/') {
15             lastsep = i;
16         }
17     }
18     ++lastsep;
19     i = 0;
20     while (path[lastsep] != 0) {
21         base[i++] = path[lastsep++];
22     }
23     base[i] = 0;
24     fputs(base, stdout);
25     return 0;
26 }
```

Basename standard inputról

basename.c (v1.0) [1–25]

```
1 /* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
2  * a fájl nevét (az útvonal nélkül).
3  *
4  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
5  * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
6  */
7
8 #include <stdio.h>
9
10 #define N 32
11
12 void basename(char path[], char base[]) {
13     int i = 0, lastsep = -1;
14     for (; path[i] != 0; ++i) {
15         if (path[i] == '/') {
16             lastsep = i;
17         }
18     }
19     ++lastsep;
20     i = 0;
21     while (path[lastsep] != 0) {
22         base[i++] = path[lastsep++];
23     }
24     base[i] = 0;
25 }
```

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]

```
1 /* Adott egy (linuxos) útvonalleírás, nyerjük ki belőle
2  * a fájl nevét (az útvonal nélkül).
3  *
4  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
5  * 2018. Szeptember 9. Gergely Tamás, gertom.inf.u-szeged.hu
6  */
7
8 #include <stdio.h>
9
10 const char * basename(const char *path) {
11     const char *base = path;
12     for (; *path != 0; ++path) {
13         if (*path == '/') {
14             base = path + 1;
15         }
16     }
17     return base;
18 }
19
20 int main(int argc, char *argv[]) {
21     if (argc < 2) {
22         return 1;
23     }
24     puts(basename(argv[1]));
25     return 0;
26 }
```


Rendezés több szempont szerint [1/8]

beszuro.c [1-25]

```
1 /* Rendezzük névsorba illetve átlag szerint a hallgatókat!
2  * Flexibilis tömbbel történik a megvalósítás, tehát a
3  * névsor hosszát nem kell előre megmondani.
4  * 1998. Február 16. Dévényi Károly, devenyi@inf.u-szeged.hu
5  * 2006. Augusztus 15. Gergely Tamás, gertom@inf.u-szeged.hu
6  * 2014. Október 15. Gergely Tamás, gertom@inf.u-szeged.hu
7  */
8
9 #include <stdio.h>
10 #include <string.h>
11 #include <stdlib.h>
12 #include <stdbool.h>
13
14 #define L 10 /* lapméret */
15
16 typedef struct { /* a tömb elem típusa */
17     char nev[21];
18     float adat;
19 } elem_t;
20
21 typedef elem_t *lap_t;
22
23 typedef lap_t *lapterkep_t;
24
25 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;
29     index_t hatar;
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 }
```

```
/* laptérkép */
/* aktuális indexhatár */
```

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 = malloc(((1 + ((n - 1) / L)) * sizeof(*a->lt)));
57         for (j = 0; j <= ((n - 1) / L); ++j) {           /* lapok létesítése */
58             a->lt[j] = malloc(L * sizeof(**a->lt));
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 [77–96]

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

Rendezés több szempont szerint [5/8]

beszuro.c [98–120]

```
98 /* Rendezés */
99
100 void beszuro_rendezes(flex_tomb_t t, rend_rel_t kisebb) {
101     /* A kisebb rendezési reláció szerinti helyben rendezés */
102     int i, j;
103     elem_t e, f;
104     for (i = 1; i < felso(t); ++i) {
105         kiolvas(t, i, &e);
106         j = i - 1;
107
108         while (true) {
109             if (j < 0) {
110                 break;
111             }
112             kiolvas(t, j, &f);
113             if (kisebb(f, e)) {
114                 break;
115             }
116             modosit(t, ((j--) + 1), f);
117         }
118         modosit(t, j + 1, e);
119     }
120 } /* beszuro_rendezes */
```

Rendezés több szempont szerint [6/8]

beszuro.c [122–145]

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

Rendezés több szempont szerint [7/8]

beszuro.c [147–163]

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



Rendezés több szempont szerint [8/8]

beszuro.c [165–179]

```
165     beszuro_rendezes(sor, rend_nev_novekvo);           /* Rend. névsor szerint */
166     printf("Névsor_szerint_rendezeve:\n");
167     kiiras(sor);                                     /* Kiíratás */
168
169     beszuro_rendezes(sor, rend_adat_novekvo);         /* Rend. adat szerint */
170     printf("Adat_szerint_rendezeve:\n");
171     kiiras(sor);                                     /* Kiíratás */
172
173     beszuro_rendezes(sor, rend_adat_csokkeno);       /* Rendezés újra */
174     printf("Adat_szerint_csokkeno_sorba_rendezeve:\n");
175     kiiras(sor);                                     /* Kiíratás */
176
177     megszuntet(& sor);                               /* a flexibilis tömb törlése */
178     return 0;
179 }
```



Adatok rendezése

beszuro.c (v1.0) [1–27]

```
1 /* Rendezzük névsorba illetve átlag szerint a hallgatókat!
2  * Flexibilis tömbbel történik a megvalósítás, tehát a
3  * névsor hosszát nem kell előre megmondani.
4  *
5  * 1998. Február 16. Dévényi Károly, devenyi@inf.u-szeged.hu
6  * 2006. Augusztus 15. Gergely Tamás, gertom@inf.u-szeged.hu
7  * 2014. Október 15. Gergely Tamás, gertom@inf.u-szeged.hu
8  * 2018. Július 24. Gergely Tamás, gertom@inf.u-szeged.hu
9  */
10
11 #include <stdio.h>
12 #include <string.h>
13 #include <stdlib.h>
14 #include <stdbool.h>
15
16 #define L 10                                     /* lapméret */
17
18 typedef struct {                                 /* a tömb elemtípusa */
19     char nev[32];
20     float adat;
21 } elem_t;
22
23 typedef elem_t *lap_t;
24
25 typedef lap_t *lapterkep_t;
26
27 typedef unsigned int index_t;
```

Adatok rendezése

beszuro.c (v1.0) [29–50]

```
29 typedef struct flex_tomb_t {
30     lapterkep_t lt;
31     index_t hatar;
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 = malloc(((1 + ((n - 1) / L)) * sizeof(*a->lt)));
57         for (j = 0; j <= ((n - 1) / L); ++j) {           /* lapok létesítése */
58             a->lt[j] = malloc(L * sizeof(**a->lt));
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) [77–96]

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

Adatok rendezése

beszuro.c (v1.0) [98–122]

```
98 /* Rendezés */
99
100 typedef bool (*rend_rel_t)(elem_t, elem_t);
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 */
```

Adatok rendezése

beszuro.c (v1.0) [124–137]

```
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 }
```



Adatok rendezése

beszuro.c (v1.0) [139–160]

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

Adatok rendezése

beszuro.c (v1.0) [162–182]

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


Előjeles/előjeltelen karaktertípus ellenőrzése [1/1]

char.c [1-14]

```
1 /* A char pontos típusának ellenőrzése.
2  * 2006. Augusztus 8. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int main() {
8     char a;
9     unsigned char b;
10    a = b = 128;
11    a >>= 1; b >>= 1;
12    printf("Ezen a gépen %signed char van\n", ((a == b) ? "un" : ""));
13    return 0;
14 }
```



Konstans változó értékének megváltoztatása [1/1]

const.c [1-14]

```
1 /* Egy const értékének megváltoztatása.
2  * 2006. Augusztus 17. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int main() {
8     const int c = 100;
9     int *p;
10    p = (int*)&c;
11    *p *= 2;
12    printf("%d\n", c);
13    return 0;
14 }
```



Csúszóátlag számítása [1/1]

csuszoatlag.c [1-25]

```
1 /* Csúszóátlag számítása fix méretű tömbön.
2  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 #define N 15
8
9 double atlag3(double v1, double v2, double v3) {
10     return (v1 + v2 + v3) / 3;
11 }
12
13 int main() {
14     double ertektomb[N], atlagtomb[N-2];
15     int i;
16     for (i = 0; i < N; ++i) {
17         printf("?_"); scanf("%lf", &(ertektomb[i]));
18     }
19     for (i = 0; i < N-2; ++i)
20         atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
21     for (i = 0; i < N-2; ++i)
22         printf("%lf;", atlagtomb[i]);
23     putchar('\n');
24     return 0;
25 }
```

Csúszoátlag számítása adott elemszámra

csuszoatlag.c (v1.0) [1–26]

```
1 /* Csúszoátlag számítása fix méretű tömbön.
2  *
3  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
4  * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 #define N 15
10
11 double atlag3(double v1, double v2, double v3) {
12     return (v1 + v2 + v3) / 3;
13 }
14
15 int main() {
16     double ertektomb[N], atlagtomb[N - 2];
17     for (int i = 0; i < N; ++i) {
18         printf("?_"); scanf("%lf", &(ertektomb[i]));
19     }
20     for (int i = 0; i < N - 2; ++i)
21         atlagtomb[i] = atlag3(ertektomb[i], ertektomb[i + 1], ertektomb[i + 2]);
22     for (int i = 0; i < N - 2; ++i)
23         printf("%lf;", atlagtomb[i]);
24     putchar('\n');
25     return 0;
26 }
```

Csúszóátlag számítás parancssorból

csuszoatlag.c (v2.0) [1-29]

```
1 /* Csúszóátlag számítása.
2  *
3  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
4  * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include <stdlib.h>
9
10 double atlag3(double v1, double v2, double v3) {
11     return (v1 + v2 + v3) / 3;
12 }
13
14 int main(int argc, char *argv[]) {
15     double erteztomb[3];
16     if (argc < 4) {
17         return 1;
18     }
19     erteztomb[0] = atof(argv[1]);
20     erteztomb[1] = atof(argv[2]);
21     for (int i = 3; i < argc; ++i) {
22         erteztomb[2] = atof(argv[i]);
23         printf("%lf;", atlag3(erteztomb[0], erteztomb[1], erteztomb[2]));
24         erteztomb[0] = erteztomb[1];
25         erteztomb[1] = erteztomb[2];
26     }
27     putchar('\n');
28     return 0;
29 }
```

Fájlkezelés text és binary módban [1/1]

data.c [1–27]

```
1 /* Fájlkezelés többféle módon.
2  * Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int main() {
8     FILE *fp, *fpb;
9     int a, b;
10
11     fp = fopen("probat.txt", "wt");
12     fprintf(fp, "%d", 34);
13     fflush(fp); fclose(fp);
14
15     fp = fopen("probat.txt", "rt");
16     fscanf(fp, "%d", &a);
17     printf("%d\n", a);
18     fclose(fp);
19
20     fpb = fopen("probab.dat", "w+b");
21     fwrite(&a, sizeof(a), 1, fpb);
22     fseek(fpb, 0, SEEK_SET);
23     fread(&b, sizeof(b), 1, fpb);
24     printf("%d\n", b);
25     fclose(fpb);
26     return 0;
27 }
```

Dátum helyességének eldöntése [1/3]

datum.c [1–12]

```
1 /* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
2  * 1997. Október 4. Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int main() {
8     int honap, nap;
9     int jo;                                /* a Boolean érték tárolására */
10
11     printf("Kérem a dátumot (hónap, nap)!\n");
12     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]

```
1 /* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
2  *   1997. Október 4.   Dévényi Károly,  devenyi@inf.u-szeged.hu
3  *   2018. Június 17.  Gergely Tamás,  gertom@inf.u-szeged.hu
4  */
5
6 #include <stdbool.h>
7 #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]

```
1 /* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
2  *   1997. Október 4.   Dévényi Károly, devenyi@inf.u-szeged.hu
3  *   2018. Június 17.  Gergely Tamás, gertom@inf.u-szeged.hu
4  */
5
6 #include <stdbool.h>
7 #include <stdio.h>
8
9 typedef struct {
10     int honap;
11     int nap;
12 } 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]

```
1 /* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
2  *   1997. Október 4.   Dévényi Károly,  devenyi@inf.u-szeged.hu
3  *   2018. Június 17.  Gergely Tamás,  gertom@inf.u-szeged.hu
4  */
5
6 #include <stdbool.h>
7 #include <stdio.h>
8
9 typedef struct {
10     int ev;
11     int honap;
12     int nap;
13 } 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]

```
1 /* Eldöntendő, hogy egy dátumként megadott számpár helyes dátum-e?
2  *   1997. Október 4.   Dévényi Károly,  devenyi@inf.u-szeged.hu
3  *   2018. Június 17.  Gergely Tamás,  gertom@inf.u-szeged.hu
4  */
5
6 #include <stdbool.h>
7 #include <stdio.h>
8
9 typedef struct {
10     int ev;
11     int honap;
12     int nap;
13 } 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]

```
1 /* Egy nap két időpontja között mennyi idő telt el.
2  * 1997. Szeptember 26. Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int main() {
8     int o1, p1;           /* az első időpont */
9     int o2, p2;           /* a második időpont */
10    int o, p;              /* az eltelt idő */
11    int k;                 /* az eltelt idő percben */
12    /* beolvasás */
13    printf("Kérem az első időpontot óra perc formában\n");
14    scanf("%d %d", &o1, &p1);
15    printf("Kérem a második időpontot óra perc formában\n");
16    scanf("%d %d", &o2, &p2);
17    /* számítás */
18    k = 60 * o2 + p2 - (60 * o1 + p1);
19    o = k / 60;
20    p = k % 60;
21    /* kiíratás */
22    printf("Az eltelt idő: %d óra %d perc.\n", o, p);
23    return 0;
24 }
```

Eltelt idő kiszámítása [1/2]

eltelt-arg.c [1–25]

```
1 /* Egy nap két időpontja között mennyi idő telt el.
2  * 2013. November 7. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7
8 typedef struct ido_t {
9     int ora;
10    int perc;
11} ido_t;
12
13 ido_t str_to_ido(const char * str) {
14    ido_t ret = {0, 0};
15    ret.ora = atoi(str);
16    for (; *str && *str != ':'; ++str);
17    if (*str) {
18        ret.perc = atoi(str + 1);
19    }
20    return ret;
21}
22
23 int ido_to_int(ido_t t) {
24    return 60 * t.ora + t.perc;
25}
```


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]

```
1 /* Egy nap két időpontja között mennyi idő telt el.
2  * 2007. Augusztus 30. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int eltelt_percek(int ora1, int perci, int ora2, int perc2) {
8     return 60 * ora2 + perc2 - (60 * ora1 + perci);
9 }
10
11 int main() {
12     int o1, p1;                /* az első időpont */
13     int o2, p2;                /* a második időpont */
14     int o, p;                  /* az eltelt idő */
15     /* beolvasás */
16     printf("Kérem az első időpontot óra perc formában\n");
17     scanf("%d%d", &o1, &p1);
18     printf("Kérem a második időpontot óra perc formában\n");
19     scanf("%d%d", &o2, &p2);
20     /* számítás */
21     o = eltelt_percek(o1, p1, o2, p2) / 60;
22     p = eltelt_percek(o1, p1, o2, p2) % 60;
23     /* kiíratás */
24     printf("Az eltelt idő: %d óra %d perc.\n", o, p);
25     return 0;
26 }
```

Eltelt idő kiszámítása [1/2]

eltelt-struct.c [1-25]

```
1 /* Egy nap két időpontja között mennyi idő telt el.
2  * 2013. November 7. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 typedef struct ido_t {
8     int ora;
9     int perc;
10 } ido_t;
11
12 int ido_to_int(ido_t t) {
13     return 60 * t.ora + t.perc;
14 }
15
16 ido_t int_to_ido(int t) {
17     ido_t ret;
18     ret.ora = t / 60;
19     ret.perc = t % 60;
20     return ret;
21 }
22
23 ido_t eltelt_ido(ido_t t1, ido_t t2) {
24     return int_to_ido(ido_to_int(t2) - ido_to_int(t1));
25 }
```

Eltelt idő kiszámítása [2/2]

eltelt-struct.c [27–39]

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



Nap két időpontja között eltelt idő

eltelt.c (v1.0) [1–19]

```
1 /* Egy nap két időpontja között mennyi idő telt el.
2  *
3  * 1997. Szeptember 26. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 void eltelt(int o1, int p1, int o2, int p2) {
10     int o, p;                /* az eltelt idő */
11     o = o2 - o1;
12     p = p2 - p1;
13     if (p < 0) {
14         o -= 1;
15         p += 60;
16     }
17     printf("%d:%02d és %d:%02d között eltelt idő %d:%02d\n",
18           o1, p1, o2, p2, o, p);
19 }
```



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]

```
1 /* Egy nap két időpontja között mennyi idő telt el.
2  *
3  * 1997. Szeptember 26. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 int eltelt(int o1, int p1, int o2, int p2) {
10     return 60 * o2 + p2 - (60 * o1 + p1);
11 }
12
13 int main() {
14     int o1, p1;                /* az első időpont */
15     int o2, p2;                /* a második időpont */
16     int k;                      /* az eltelt idő */
17     /* beolvasás */
18     printf("Kérem az első időpontot óra:perc formában\n");
19     scanf("%d_%d", &o1, &p1);
20     printf("Kérem a második időpontot óra:perc formában\n");
21     scanf("%d_%d", &o2, &p2);
22     k = eltelt(o1, p1, o2, p2);
23     printf("%d:%02d_és_%d:%02d között eltelt idő %d:%02d\n",
24           o1, p1, o2, p2, k / 60, k % 60);
25     return 0;
26 }
```

Nap két időpontja között eltelt idő

eltelt.c (v3.0) [1–27]

```
1 /* Egy nap két időpontja között mennyi idő telt el.
2  *
3  * 2013. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 typedef struct ido_t {
10     int ora;
11     int perc;
12 } ido_t;
13
14 int ido_to_int(ido_t t) {
15     return 60 * t.ora + t.perc;
16 }
17
18 ido_t int_to_ido(int t) {
19     ido_t ret = {0, 0};
20     ret.ora = t / 60;
21     ret.perc = t % 60;
22     return ret;
23 }
24
25 ido_t eltelt_ido(ido_t t1, ido_t t2) {
26     return int_to_ido(ido_to_int(t2) - ido_to_int(t1));
27 }
```


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]

```
1 /* Egy nap két időpontja között mennyi idő telt el.
2  *
3  * 2013. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include <stdlib.h>
9
10 typedef struct ido {
11     int ora;
12     int perc;
13 } ido;
14
15 ido str_to_ido(const char * str) {
16     ido ret = {0, 0};
17     ret.ora = atoi(str);
18     for (; *str && *str != ':'; ++str);
19     if (*str) {
20         ret.perc = atoi(++str);
21     }
22     return ret;
23 }
24
25 int ido_to_int(ido t) {
26     return 60 * t.ora + t.perc;
27 }
```

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]

```
1 /* Egy nap két időpontja között mennyi idő telt el.
2  *
3  * 2013. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include <stdlib.h>
9
10 typedef struct ido {
11     int ora;
12     int perc;
13     int masodperc;
14 } ido;
15
16 ido str_to_ido(const char * str) {
17     ido ret = {0, 0, 0};
18     ret.ora = atoi(str);
19     for (; *str && *str != ':'; ++str);
20     if (*str) {
21         ret.perc = atoi(++str);
22     }
23     for (; *str && *str != ':'; ++str);
24     if (*str) {
25         ret.masodperc = atoi(++str);
26     }
27     return ret;
28 }
```

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:%02dés %d:%02d:%02d közötteltelt időidő %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 bájtrendjének meghatározása [1/1]

endian.c [1–16]

```
1 /* Eldönti, hogy milyen a bájtrend a számítógépen.
2  * 1998. Április 24. Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int main() {
8     int x = 1;
9
10    if (*(char *)&x == 1) {
11        printf("little-endian\n");
12    } else {
13        printf("big-endian\n");
14    }
15    return 0;
16 }
```



A nappalis gyakorlat értékelése [1/10]

eredmeny.c [1-23]

```
1 /* Programozás alapjai gyakorlat értékelés.
2  *
3  * A program a parancssorban sorban megadott zh és plusz pontok alapján
4  * kiszámolja a várható érdemjegyet.
5  *
6  * 2023. Május 22. Gergely Tamás, gertom@inf.u-szeged.hu
7  */
8
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <stdbool.h>
12 #include <string.h>
13
14 #ifndef SILENT
15 #define GR "A gyakorlat_eredménye: "
16 #else
17 #define GR
18 #endif
19 #define G1 "elégtelen (1) "
20 #define G2 "elégséges (2) "
21 #define G3 "közepes (3) "
22 #define G4 "jó (4) "
23 #define G5 "jeles (5) "
```

A nappalis gyakorlat értékelése [2/10]

eredmeny.c [25–43]

```
25#define MAX_ZH_PER_WEEK 2
26
27#define OPT_HELP_S "-h"
28#define OPT_HELP_L "--help"
29#define OPT_NO_MIN "--no-minimums"
30
31typedef struct {
32    int    min;
33    char  *jegy;
34} eredmeny_t;
35
36const eredmeny_t G[] = {
37    {80, G5},
38    {70, G4},
39    {57, G3},
40    {45, G2},
41    { 0, G1},
42    { 0, NULL}
43};
```



A nappalis gyakorlat értékelése [3/10]

eredmeny.c [45–61]

```
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 };
48 const int MAXPLUS = 5;
49 const int MAXPRGP = 5;
50 const int MAXPSUM = 10;
51 const int NINCSMIN = 1;
52 const int NINCSVED = 1;
53
54 bool minimum_ellenorzes = true;
55 int vedes_nem_sikerult = 0;
56
57 int heti_pont(int w, int zhn, int zhp[], int hfp[]) {
58     int sum = 0;
59     int db = ZHPWEEK[w];
60     bool zhnemved = false;
61     bool hfnemved = false;
```



A nappalis gyakorlat értékelése [4/10]

eredmeny.c [63–90]

```
63     for (int i = 0, j = 0; i < db; ++i, ++zhn) {
64         if (zhp[i] < 0) {
65             zhnemved = true;
66             zhp[i] = 0;
67         }
68         if (hfp[i] < 0) {
69             hfnemved = true;
70             hfp[i] = 0;
71         }
72         sum += ((zhp[i] >= ZHMAXPT[zhn]) ? ZHMAXPT[zhn] : zhp[i]) + (!!hfp[j++]);
73     }
74
75     if (zhnemved) {
76 #ifndef SILENT
77         printf("Figyelmeztetés! A(z) %d. zh védése nincs elfogadva.\n", w + 1);
78 #endif
79         ++vedes_nem_sikerult;
80     }
81
82     if (hfnemved) {
83 #ifndef SILENT
84         printf("Figyelmeztetés! A(z) %d. házi feladat védése nincs elfogadva.\n", w + 1);
85 #endif
86         ++vedes_nem_sikerult;
87     }
88
89     return sum;
90 }
```

A nappalis gyakorlat értékelése [5/10]

eredmeny.c [92–110]

```
92 bool min_bukas(int w, int zhp) {
93     static int nincs_min = 0;
94     if (minimum_ellenorzes && zhp < MINIMUM[w]) {
95 #ifndef SILENT
96         printf("Figyelmeztetés! A(z) %d. zh után nincs meg a minimális pontszám.\n"
97              "    elvárt: %d, meglévő: %d.\n", w + 1, MINIMUM[w], zhp);
98 #endif
99         if (++nincs_min > NINCSMIN) {
100 #ifndef SILENT
101             printf("A minimum pontszám kritériumait legalább %d alkalommal "
102                  "nem teljesítette, ami több, mint az elfogadható %d alkalom.\n",
103                   nincs_min, NINCSMIN);
104 #endif
105             printf(GR G1 "\n");
106             return true;
107         }
108     }
109     return false;
110 }
```



A nappalis gyakorlat értékelése [6/10]

eredmeny.c [112–135]

```
112 void usage(const char* progname) {
113     int db;
114     int maxpont = 0;
115     printf("Használat: %s [\" OPT_HELP_S \"|\" OPT_HELP_L \"|\" OPT_NO_MIN \"]", progname);
116     for (int w = 0; (db = ZHPWEEK[w]); ++w) {
117         if (db == 1) {
118             printf("_hf%d_zh%d", w + 1, w + 1);
119         } else {
120             for (int i = 0; i < db; ++i) {
121                 printf("_hf%d/%d_zh%d/%d", w + 1, i + 1, w + 1, i + 1);
122             }
123         }
124     }
125     printf("_plusz_pprog\n"
126           "\n"
127           "UUUUhfN: _az_N_ _heti_hazi_feladat_pontjai\n"
128           "UUUUhfN/M: _az_N_ _heti_zh_M_ _hazi_feladat_pontjai\n"
129           "UUUUzhN: _az_N_ _heti_zh_pontszama\n"
130           "UUUUzhN/M: _az_N_ _heti_zh_M_ _reszpontszama\n"
131           "UUUUplusz: _plusz_pontszam\n"
132           "UUUUpprog: _plusz_program_pontszam\n"
133           "\n"
134           "Összpontszámok_minimuma_(a_plusz_és_pprog_nem_számitanak_bele):\n"
135           "\n");
```

A nappalis gyakorlat értékelése [7/10]

eredmeny.c [136–161]

```
136     for (int w = 0, tmp = 0; (db = ZHPWEEK[w]); ++w) {
137         maxpont += db;
138         for (int i = 0; i < db; ++i) {
139             maxpont += ZHMAXPT[tmp++];
140         }
141         printf("%%%%d. hét után %d pont (az addigi összesen %d pontból)\n",
142             w + 1, MINIMUM[w], maxpont);
143     }
144     maxpont += MAXPSUM;
145
146     printf("\n"
147         "Ha a minimum pontszámot több, mint %d alkalommal"
148         " nem sikerül teljesíteni,\n"
149         " vagy a leadott feladatok közül több, mint %d darabot"
150         " nem sikerül megvédeni,\n"
151         " akkor a félév végeredménye minden másról függetlenül" G1 ".\n"
152         "\n"
153         "Értékelés, az összes részpont, plusz (maximum %d)"
154         " és a progra (maximum %d, összesen max %d) összege alapján:\n"
155         "\n",
156         NINCSMIN, NINCSVED, MAXPLUS, MAXPRGP, MAXPSUM);
157     for (const eredmény_t *p = G; p->jegy; ++p) {
158         printf("%%%%3d -- %3d pont: %s\n", p->min, maxpont, p->jegy);
159         maxpont = p->min - 1;
160     }
161 }
```

A nappalis gyakorlat értékelése [8/10]

eredmeny.c [163–186]

```
163 int main(int argc, char *argv[]) {
164     int sum = 0;
165     int plus = 0;
166     int prgp = 0;
167     int idx = 1;
168     int db;
169     int w, zhn;
170
171     if (argc < 2) {
172 #ifndef SILENT
173         usage(argv[0]);
174 #endif
175         return 1;
176     }
177
178     if (!strcmp(OPT_HELP_S, argv[1]) || !strcmp(OPT_HELP_L, argv[1])) {
179         usage(argv[0]);
180         return 0;
181     }
182
183     if (!strcmp(OPT_NO_MIN, argv[1])) {
184         minimum_ellenorzes = false;
185         idx = 2;
186     }
```

A nappalis gyakorlat értékelése [9/10]

eredmeny.c [188–215]

```
188     for (w = 0, zhn = 0; (db = ZHPWEEK[w]) && (idx < argc); ++w, zhn += db) {
189         int hfp[MAX_ZH_PER_WEEK];
190         int zhp[MAX_ZH_PER_WEEK];
191         for (int i = 0; i < db; ++i) {
192             hfp[i] = (idx < argc) ? atoi(argv[idx]) : 0;
193             ++idx;
194             zhp[i] = (idx < argc) ? atoi(argv[idx]) : 0;
195             ++idx;
196         }
197         if (idx <= argc) {
198             if (min_bukas(w, sum += heti_pont(w, zhn, zhp, hfp))) {
199                 return 0;
200             }
201         } else {
202             break;
203         }
204     }
205
206     if (vedes_nem_sikerult > NINCSVED) {
207 #ifndef SILENT
208         printf("A leadott feladata %d alkalommal nem lett elfogadva, ami több,"
209             " mint az elfogadható alkalom.\n"
210             "A(z) %d pontját így nem tudjuk figyelembe venni.\n",
211             vedes_nem_sikerult, NINCSVED, sum);
212 #endif
213         printf(GR G1 "\n");
214         return 0;
215     }
```

A nappalis gyakorlat értékelése [10/10]

eredmeny.c [217–245]

```
217     if (db > 0) { // Ha még nincs pontszám minden zh-hoz (break volt)
218 #ifndef SILENT
219     printf("%d zh eredménye alapján jelenleg %d pontja van.\n", w, sum);
220 #endif
221     return 0;
222 }
223
224 if (idx < argc) { // ha van plusz pont is
225     plus = atoi(argv[idx++]);
226 }
227 plus = (plus > MAXPLUS) ? MAXPLUS : plus;
228 if (idx < argc) { // ha van program plusz pont is
229     prgp = atoi(argv[idx++]);
230 }
231 plus += (prgp > MAXPRGP) ? MAXPRGP : prgp;
232
233 sum += (plus > MAXPSUM) ? MAXPSUM : plus;
234
235 #ifndef SILENT
236 printf("A végső pontszáma %s %d.\n", (idx < argc) ? "legalább" : "", sum);
237 #endif
238 for (const eredmény_t *p = G; p->jegy; ++p) { // A pontszámhoz tartozó jegy keresése
239     if (sum >= p->min) {
240         printf(GR "%s\n", p->jegy);
241         return 0;
242     }
243 }
244 return -1;
245 }
```


Egy dátum az év hányadik napja? [1/2]

evnapja.c [1–11]

```
1 /* Egy YYYY-MM-DD formátumban megadott dátum az év hányadik napja?
2  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 typedef struct {
8     int ev;
9     int ho;
10    int nap;
11} datum_t;
```



Egy dátum az év hányadik napja? [2/2]

evnapja.c [13–36]

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

Egy dátum az év hányadik napja?

evnapja.c (v1.0) [1–13]

```
1 /* Egy YYYY-MM-DD formátumban megadott dátum az év hányadik napja?
2  *
3  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
4  * 2018. Július 24. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 typedef struct {
10     int ev;
11     int ho;
12     int nap;
13 } 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]

```
1 /* Közelítő integrálás a trapéz szabály segítségével.
2  * Az integrálandó függvény nevét a parancssorból kapjuk.
3  * Meg kell adnunk az intervallumot is és a finomságot is.
4  * Kérhető help is.
5  * 1998. Április 14. Dévényi Károly, devenyi@inf.u-szeged.hu
6  * 2020. Július 24. Gergely Tamás, gertom@inf.u-szeged.hu
7  */
8
9 #include <stdio.h>
10 #include <stdlib.h>
11 #include <string.h>
12 #include <math.h>
13 #include <errno.h>
14
15 typedef struct tablaelem_t {
16     char *nev;                               /* A függvény neve szövegesen */
17     double (*fuggveny)();                   /* A függvényre mutató pointer */
18 } tablaelem_t;
19
20 static double help();
21
22 static double expx();
23
24 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     { "exp",  exp  },
31     { "sin2x", sin2x },
32     { "help", help },
33     { NULL,   NULL }
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 }
```

/ A táblázat végét jelzi */*



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 -> fuggvény)(a), /* A függvény értéke a-nál */
107           argv[1],             /* Pointer a függvény nevére */
108           argv[3],             /* Pointer a második számra */
109           (*tp -> fuggvény)(b), /* A függvény értéke b-nél */
110           argv[1],             /* Pointer a függvény nevére */
111           trapez(*tp -> fuggvény, 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
4 typedef ???          ft_elem_t;          /* a tömb elemtípusa */
5 typedef unsigned int ft_index_t;
6 typedef struct flex_tomb_t {
7     ft_elem_t *tomb;                    /* tömb */
8     ft_index_t hatar;                   /* aktuális indexhatár */
9 } flex_tomb_t;
10
11 /* A műveletek deklarációja: */
12 void ft_kiolvas(flex_tomb_t a, ft_index_t i, ft_elem_t *x);
13 void ft_modosit(flex_tomb_t a, ft_index_t i, ft_elem_t x);
14 index_t ft_felső(flex_tomb_t a);
15 void ft_letesit(flex_tomb_t *a, ft_index_t n);
16 void ft_megszuntet(flex_tomb_t *a);
17 void ft_novel(flex_tomb_t *a, ft_index_t d);
18 void ft_csokkent(flex_tomb_t *a, ft_index_t d);
19
20 #endif // FTOMB_SIMPLE_H
```

Flexibilis tömb egyszerű megvalósítás [2/3]

ftomb_simple_t.c [1-18]

```
1#include <stdlib.h>
2#include "ftomb_simple_t.h"
3
4void ft_kiolvas(flex_tomb_t a, ft_index_t i, ft_elem_t *x) {
5    if (i < a.hatar) {
6        *x = a.tomb[i];
7    }
8}
9
10void ft_modosit(flex_tomb_t a, ft_index_t i, ft_elem_t x) {
11    if (i < a.hatar) {
12        a.tomb[i] = x;
13    }
14}
15
16index_t ft_felso(flex_tomb_t a) {
17    return a.hatar;
18}
```



Flexibilis tömb egyszerű megvalósítás [3/3]

ftomb_simple_t.c [20–37]

```
20 void ft_letesit(flex_tomb_t *a, ft_index_t n) {
21     a->hatar = n;
22     a->tomb = ((n) ? malloc(n * sizeof(*a->tomb)) : NULL);
23 }
24
25 void ft_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 ft_novel(flex_tomb_t *a, ft_index_t d) {
32     a->tomb = realloc(a->tomb, (a->hatar += d) * sizeof(*a->tomb));
33 }
34
35 void ft_csokkent(flex_tomb_t *a, ft_index_t d) {
36     a->tomb = realloc(a->tomb, (a->hatar -= d) * sizeof(*a->tomb));
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
6 typedef ???          ft_elem_t;             /* a tömb elemtípusa */
7 typedef unsigned int ft_index_t;
8 typedef ft_elem_t *ft_lap_t;
9 typedef ft_lap_t *ft_lapterkep_t;
10 typedef struct flex_tomb_t {
11     ft_lapterkep_t lt;                       /* lapterkép */
12     ft_index_t    hatar;                    /* aktuális indexhatár */
13 } flex_tomb_t;
14
15 /* A műveletek deklarációja: */
16 void ft_kiolvas(flex_tomb_t a, ft_index_t i, ft_elem_t *x);
17 void ft_modosit(flex_tomb_t a, ft_index_t i, ft_elem_t x);
18 index_t ft_felso(flex_tomb_t a);
19 void ft_letesit(flex_tomb_t *a, ft_index_t n);
20 void ft_megszuntet(flex_tomb_t *a);
21 void ft_novel(flex_tomb_t *a, ft_index_t d);
22 void ft_csokkent(flex_tomb_t *a, ft_index_t d);
23
24 #endif // FTOMB_H
```

Flexibilis tömb megvalósítás [2/4]

ftomb_t.c [1-18]

```
1#include <stdlib.h>
2#include "ftomb_t.h"
3
4void ft_kiolvas(flex_tomb_t a, ft_index_t i, ft_elem_t *x) {
5    if (i < a.hatar) {
6        *x = a.lt[i / L][i % L];
7    }
8}
9
10void ft_modosit(flex_tomb_t a, ft_index_t i, ft_elem_t x) {
11    if (i < a.hatar) {
12        a.lt[i / L][i % L] = x;
13    }
14}
15
16index_t ft_felso(flex_tomb_t a) {
17    return a.hatar;
18}
```



Flexibilis tömb megvalósítás [3/4]

ftomb_t.c [20-43]

```
20 void ft_letesit(flex_tomb_t *a, ft_index_t n) {
21     a->hatar = n;
22     if (n) {
23         int j;
24         a->lt = malloc(((1 + ((n - 1) / L)) * sizeof(*a->lt)));
25         for (j = 0; j <= ((n - 1) / L); ++j) {           /* lapok létesítése */
26             a->lt[j] = malloc(L * sizeof(**a->lt));
27         }
28     } else {
29         a->lt = NULL;
30     }
31 }
32
33 void ft_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 [45–64]

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


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]

```
1 /* Gráf megvalósítása típuselrejtéssel.
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 #include <stdlib.h>
7 #include "graf.h"
8
9 typedef struct _graf_t {
10     int    n;
11     char *mx;
12 } _graf_t;
13
14 static bool jo_pont(_graf_t *g, pont_t p) {
15     return 0 <= p && p < g->n;
16 }
```



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

graf1.c [18–39]

```
18 graf_t gr_letesit(int n) {
19     _graf_t *ptr;
20     ptr = malloc(sizeof(*ptr));
21     if (ptr) {
22         ptr->n = n;
23         ptr->mx = malloc(n * n * sizeof(*ptr->mx));
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]

```
1 /* Gráf megvalósítása típuselrejtéssel.
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 #include <stdlib.h>
7 #include "graf.h"
8
9 typedef struct _graf_t {
10     int    n;
11     int    *be;
12     int    *ki;
13     pont_t *mx;
14 } _graf_t;
15
16 static bool jo_pont(_graf_t *g, pont_t p) {
17     return 0 <= p && p < g->n;
18 }
```



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

graf2.c [20–44]

```
20 graf_t gr_letesit(int n) {
21     _graf_t *ptr;
22     ptr = malloc(sizeof(*ptr));
23     if (ptr) {
24         ptr->n = n;
25         ptr->be = malloc(n * sizeof(*ptr->be));
26         ptr->ki = malloc(n * sizeof(*ptr->ki));
27         ptr->mx = malloc(n * n * sizeof(*ptr->mx));
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(c1,cg) } PRINT(c1,cg) }
12 #define ISM_A      5000000
13 #define ISM_B      2000000
14 #define ISM_C      1000000
15 #define ISM_D      500000
16 #define ISM_E      200000
17 #define ISM_F      100000
18 #define ISM_G      50000
19 #define ISM_H      20000
20 #define ISM_I      10000
21 #define ISM_J      5000
22 #define ISM_K      2000
23 #define ISM_L      1000
24 #define ELEK      100
25 #define START(c)   c = time(NULL);
26 #define PRINT(c1,cg) { long now = time(NULL); \
27     printf("Eltelet␣idő:␣%31d␣(%31d)␣másodperc\n", now - c1, 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  * preproceszor 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(halmaz_kishalmaz_t))
13 #define HALMAZELEM_PRINT_FORMAT "lld"
14
15 typedef long long int halmaz_kishalmaz_t;
16
17 typedef long long int halmaz_elem_t;
18
19 typedef struct {
20     halmaz_elem_t      n;          /* Az univerzum a [0..N) */
21     long long int      m;          /* A kishalmazok száma */
22     halmaz_kishalmaz_t *tar;      /* A kishalmazok tömbje */
23 } halmaz_t;
```

Halmaz modul interfész [2/2]

halmaz.h [25–39]

```
25 halmaz_t halmaz_letesit(halmaz_elem_t);
26 void halmaz_megszuntet(halmaz_t*);
27 void halmaz_uresit(halmaz_t);
28 void halmaz_bovit(halmaz_t, halmaz_elem_t);
29 void halmaz_torol(halmaz_t, halmaz_elem_t);
30 bool halmaz_eleme(halmaz_t, halmaz_elem_t);
31 void halmaz_egyesites(halmaz_t, halmaz_t, halmaz_t);
32 void halmaz_metszet(halmaz_t, halmaz_t, halmaz_t);
33 void halmaz_kulonbseg(halmaz_t, halmaz_t, halmaz_t);
34 bool halmaz_egyenlo(halmaz_t, halmaz_t);
35 bool halmaz_resz(halmaz_t, halmaz_t);
36 bool halmaz_ures_e(halmaz_t);
37 void halmaz_ertekadas(halmaz_t, halmaz_t);
38
39 #endif /* HALMAZ_H */
```



Halmaz modul implementáció [1/4]

halmaz.c [1–25]

```
1 /* A halmaz modul függvényeinek megvalósítása.
2  * 2006. Augusztus 16. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <stdlib.h>
6 #include "halmaz.h"
7
8 halmaz_t halmaz_letesit(halmaz_elem_t n) {
9     halmaz_t ret;
10    ret.n = n;
11    if (n) {
12        ret.m = (n - 1) / K + 1;
13        ret.tar = malloc(ret.m * sizeof(*ret.tar));
14    } else {
15        ret.m = 0;
16        ret.tar = NULL;
17    }
18    return ret;
19 }
20
21 void halmaz_megszuntet(halmaz_t *h) {
22    free(h->tar);
23    h->n = h->m = 0;
24    h->tar = NULL;
25 }
```

Halmaz modul implementáció [2/4]

halmaz.c [27–49]

```
27 void halmaz_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 halmaz_bovit(halmaz_t h, halmaz_elem_t x) {
35     if (x < h.n) {
36         h.tar[x / K] |= (1LL << (x % K));
37     }
38 }
39
40 void halmaz_torol(halmaz_t h, halmaz_elem_t x) {
41     if (x < h.n) {
42         h.tar[x / K] &= ~(1LL << (x % K));
43     }
44 }
45
46 bool halmaz_eleme(halmaz_t h, halmaz_elem_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 halmaz_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 halmaz_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 halmaz_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 halmaz_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 halmaz_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 halmaz_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 halmaz_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 HALMAZ_K (8*sizeof(halmaz_kishalmaz))
7 #define HALMAZ_M ???
8 #define HALMAZ_H_MAX (HALMAZ_M*HALMAZ_K)
9
10 typedef unsigned long int halmaz_elem_t;
11 typedef halmaz_elem_t halmaz_kishalmaz_t;
12 typedef halmaz_kishalmaz_t halmaz_t[HALMAZ_M];
13
14 void halmaz_uresit(halmaz_t);
15 void halmaz_bovit(halmaz_t, halmaz_elem_t);
16 void halmaz_torol(halmaz_t, halmaz_elem_t);
17 bool halmaz_eleme(halmaz_t, halmaz_elem_t);
18 void halmaz_egyesites(halmaz_t h1, halmaz_t h2, halmaz_t h);
19 void halmaz_metszet(halmaz_t h1, halmaz_t h2, halmaz_t h);
20 void halmaz_kulonbseg(halmaz_t h1, halmaz_t h2, halmaz_t h);
21 bool halmaz_egyenlo(halmaz_t h1, halmaz_t h2);
22 bool halmaz_resz(halmaz_t h1, halmaz_t h2);
23 bool halmaz_ures_e(halmaz_t h);
24 void halmaz_ertekadas(halmaz_t h1, halmaz_t h2);
25
26 #endif // HALMAZ_H
```

Halmaz megvalósítás [2/4]

halmaz_t.c [1-21]

```
1#include "halmaz_t.h"
2
3void halmaz_uresit(halmaz_t h) {
4    long int i;
5    for (i = 0; i < HALMAZ_M; ++i)
6        h[i] = 0;
7}
8
9void halmaz_bovit(halmaz_t h, halmaz_elem_t x) {
10    if (x < HALMAZ_H_MAX)
11        h[x / HALMAZ_K] |= (1 << (x % HALMAZ_K));
12}
13
14void halmaz_torol(halmaz_t h, halmaz_elem_t x) {
15    if (x < HALMAZ_H_MAX)
16        h[x / HALMAZ_K] &= ~(1 << (x % HALMAZ_K));
17}
18
19bool halmaz_eleme(halmaz_elem_t x, halmaz_t h) {
20    return (x < HALMAZ_H_MAX) && (h[x / HALMAZ_K] & (1 << (x % HALMAZ_K)));
21}
```


Halmaz megvalósítás [3/4]

halmaz_t.c [23–45]

```
23 void halmaz_egyesites(halmaz_t h1, halmaz_t h2, halmaz_t h) {
24     long int i;
25     for (i = 0; i < HALMAZ_M; ++i)
26         h[i] = h1[i] | h2[i];
27 }
28
29 void halmaz_metszet(halmaz_t h1, halmaz_t h2, halmaz_t h) {
30     long int i;
31     for (i = 0; i < HALMAZ_M; ++i)
32         h[i] = h1[i] & h2[i];
33 }
34
35 void halmaz_kulonbseg(halmaz_t h1, halmaz_t h2, halmaz_t h) {
36     long int i;
37     for (i = 0; i < HALMAZ_M; ++i)
38         h[i] = h1[i] & ~(h2[i]);
39 }
40
41 bool halmaz_egyenlo(halmaz_t h1, halmaz_t h2) {
42     long int i;
43     for (i = 0; (i < HALMAZ_M) && (h1[i] == h2[i]); ++i);
44     return i == HALMAZ_M;
45 }
```

Halmaz megvalósítás [4/4]

halmaz_t.c [47–63]

```
47 bool halmaz_resz(halmaz_t h1, halmaz_t h2) {
48     long int i;
49     for (i = 0; (i < HALMAZ_M) && !(h1[i] & ~(h2[i])); ++i);
50     return i == HALMAZ_M;
51 }
52
53 bool halmaz_urese(halmaz_t h) {
54     long int i;
55     for (i = 0; (i < HALMAZ_M) && !(h[i]); ++i);
56     return i == HALMAZ_M;
57 }
58
59 void halmaz_ertekadas(halmaz_t h1, halmaz_t h2) {
60     long int i;
61     for (i = 0; i < HALMAZ_M; ++i)
62         h1[i] = h2[i];
63 }
```



Hanoi tornyai [1/2]

hanoi.c [1–21]

```
1 /* A Hanoi tornyai játék megvalósítása rekurzív eljárással.
2  * 1997. Október 31. Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 void mozgat(int innen, int ide) {          /* Átrak egy korongot innen ide */
8     printf("␣Tegyük␣át␣egy␣korongot␣a␣%d.␣oszlopról␣a␣%d.␣oszlopra!\n", innen, ide);
9 }
10
11 void hanoi(int n,                          /* ilyen magas a torony */
12            int rol,                        /* erről a toronyról */
13            int ra) {                       /* erre a toronyra */
14     if (n == 1) {
15         mozgat(rol, ra);
16     } else {
17         hanoi(n - 1, rol, 6 - ra - rol);
18         mozgat(rol, ra);
19         hanoi(n - 1, 6 - ra - rol, ra);
20     }
21 }
```

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]

```
1 /* Milyen háromszöget határoz meg három pozitív valós szám,  
2  * mint a háromszög három oldalhosszúsága?  
3  * 1997. Október 13. Dévényi Károly, devenyi@inf.u-szeged.hu  
4  */  
5  
6 #include <stdio.h>  
7  
8 int main() {  
9     double a, b, c;    /* a háromszög oldalhosszúságai */  
10    double m;         /* munkaváltozó a cseréhez */  
11  
12    printf("Kérem a három pozitív valós számot!\n");  
13    scanf("%lf%lf%lf", &a, &b, &c);  
14  
15                                /* a, b, c átrendezése úgy, hogy a>=b, c legyen */  
16    if (a < b) {                /* a és b átrendezése */  
17        m = a;  
18        a = b;  
19        b = m;  
20    }  
21    if (a < c) {                /* a és c átrendezése */  
22        m = a;  
23        a = c;  
24        c = m;  
25    }
```

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("Egyenlőszárú derékszögű háromszög.\n");
36         } else {
37             printf("Egyenlőszárú 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]

```
1 /* Milyen háromszöget határoz meg három pozitív valós szám,  
2 * mint a háromszög három oldalhosszúsága? A valós típus  
3 * pontatlansága miatt toleranciával számolunk.  
4 * 1997. Október 13. Dévényi Károly, devenyi@inf.u-szeged.hu  
5 * 2014. Február 19. Gergely Tamás, gertom@inf.u-szeged.hu  
6 */  
7  
8 #include <stdio.h>  
9  
10 #define EQUALS(X,Y) ( ( ((X) > (Y)) ? ((X) - (Y)) : ((Y) - (X)) ) <= 1e-10 )  
11  
12 int main() {  
13     double a, b, c;    /* a háromszög oldalhosszúságai */  
14     double m;         /* munkaváltozó a cseréhez */  
15  
16     printf("Kérem a három pozitív valós számot!\n");  
17     scanf("%lf%lf%lf", &a, &b, &c);  
18  
19     /* a,b,c átrendezése úgy, hogy a>=b,c legyen */  
20     if (a < b) {      /* a és b átrendezése */  
21         m = a;  
22         a = b;  
23         b = m;  
24     }  
25     if (a < c) {      /* a és c átrendezése */  
26         m = a;  
27         a = c;  
28         c = m;  
29     }
```

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]

```
1 /* Hellő Világ!  
2  * 2004. November 3.  Gergely Tamás, gertom@inf.u-szeged.hu  
3  */  
4  
5 #include <stdio.h>  
6  
7 int main() {  
8     printf("Hellő Világ!\n");  
9     return 0;  
10 }
```



Határozott integrál kiszámítása [1/3]

integral.c [1-24]

```
1 /* Közelítő integrálás a trapéz szabály segítségével.
2  * Az integrálandó függvényt paraméterként kapjuk.
3  * 1998. Április 14. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2006. Augusztus 14. Gergely Tamás, gertom@inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include <math.h>
9
10 /* Az integrálandó függvény típusa */
11 typedef double (*fuggveny_t)(double x);
12
13 static double trapez(fuggveny_t f,          /* f(x)-t integráljuk az */
14                    double a, double b,    /* a,b intervallumon */
15                    int n) {               /* n részre osztva az interv.-t */
16     /* Közelítő integrálás a trapéz szabály segítségével. */
17     const double h = (b - a) / n;
18     double area = 0.0;
19     int i;                                /* a ciklusváltozó */
20
21     for (i = 1; i < n; ++i) {            /* fgv. értékek összegzése */
22         area += (*f)(a + i * h);         /* f(a + i * h) */
23     }
24     return (area * h + ((*f)(a) + (*f)(b)) / 2.0 * h); /* (f(a) + f(b)) */
25 }
```

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("%10.5lf\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("%10.5lf\n", trapez(sin2x, a, b, n));
57     return 0;
58 }
```

Kamatos kamat számítása [1/2]

kamatos.c [1-22]

```
1 /* Kamatos-kamat számítás a hatványozás függvény segítségével.
2  * 1997. Október 31. Dévényi Károly, devenyi@inf.u-szeged.hu
3  * 2014. Szeptember 23. Gergely Tamás, gertom@inf.u-szeged.hu
4  */
5
6 #include <stdio.h>
7 #include <math.h>
8
9 double hatvany(double x, int n) {
10     /* x n-edik hatványát kiszámító függvény */
11     if (n == 0) {
12         return 1.0;
13     } else if (x == 0.0) {
14         return 0.0;
15     } else if (x > 0.0) {
16         return exp(n * log(x));
17     } else if (n & 1) {
18         return -exp(n * log(-x));
19     } else {
20         return exp(n * log(-x));
21     }
22 }
```

Kamatos kamat számítása [2/2]

kamatos.c [24–41]

```
24 int main() {
25     double osszeg, uj_osszeg;
26     int kamatlab, ev;
27
28     printf("A kamatozó összeg? ");
29     scanf("%lg%*[\n]", &osszeg);
30     getchar();
31     printf("A kamatláb? ");
32     scanf("%d%*[\n]", &kamatlab);
33     getchar();
34     printf("A kamatozási évek száma? ");
35     scanf("%d%*[\n]", &ev);
36     getchar();
37     uj_osszeg = osszeg * hatvany(1.0 + kamatlab / 100.0, ev);
38     printf("A kamatos kamattal növelt összeg:");
39     printf("%10.2lf\n", uj_osszeg);
40     return 0;
41 }
```



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];
15 int i;                                /* innen fogunk írni */
                                        /* 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
27     for (i = 0; i < DBSZAM; ++i) {        /* a tomb feltöltése jól látható értékekkel */
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]

```
1 /* A beolvasott R relációt tartalmazó legszűkebb ekvivalencia
2    reláció szerinti osztályozást határozzuk meg.
3    Vázlat, nem fordítható C program.
4    2006. Augusztus 14. Gergely Tamás, gertom@inf.u-szeged.hu
5 */
6 #include <stdio.h>
7 #define N 10 /* maximális elemszám */
8 typedef Halmaz(U) Halmaz; /* EZ ÍGY NEM C !!! */
9
10 int main() {
11     Halmaz H[N];
12     unsigned short i, j, ti, tj;
13     for (i = 1; i <= N; ++i) { /* inicializálás */
14         Uresit(H[i - 1]); Bovit(H[i - 1], i);
15     }
16     printf("Kérem a relációban lévő számpárokat!\n");
17     scanf("%hd%hd%*[\n]", &i, &j);
18     getchar();
19     while (i != 0) {
20         for (ti = 0; !Eleme(i, H[ti]); ++ti); /* amelyik H[ti] halmazban van i ? */
21         for (tj = 0; !Eleme(j, H[tj]); ++tj); /* amelyik H[tj] halmazban van j ? */
22         if (ti != tj) { /* H[ti] és H[tj] összevonása */
23             Egyesites(H[ti], H[tj], H[ti]);
24             Uresit(H[tj]);
25         }
26         scanf("%hd%hd%*[\n]", &i, &j);
27         getchar();
28     }
```

Klikkek meghatározása absztrakt Halmaz típussal [2/2]

klikk-abstract.c [30–40]

```
30 printf("Az osztályok:\n");
31 for (i = 1; i <= N; ++i) {
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 }
```

/ az osztályok kiírása */*



Klikkek meghatározása [1/2]

klikk.c [1–19]

```
1 /* A beolvasott R relációt tartalmazó legszűkebb ekvivalencia
2  * reláció szerinti osztályozást határozzuk meg.
3  * 1997. December 6. Dévényi Károly, devenyi@inf.u-szeged.hu
4  */
5
6 #include <stdio.h>
7
8 #define N 10 /* maximális elemszám */
9
10 typedef unsigned short int univerzum_t; /* az univerzum */
11 typedef long int halmaz_t; /* N kicsi, ezért elegendő egy long int */
12
13 int main(int argc, char *argv[]) {
14     halmaz_t h[N];
15     univerzum_t i, j, ti, tj;
16
17     for (i = 1; i <= N; ++i) { /* inicializálás */
18         h[i - 1] = 11 << i;
19     }
```



Klikkek meghatározása [2/2]

klikk.c [21–46]

```
21 printf("Kérem a relációban lévő 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; ((11 << i) & h[ti]) == 0; ++ti);
26     /* a j-t tartalmazó halmaz tj indexének keresése */
27     for (tj = 0; ((11 << 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 kiírása */
37     if (h[i - 1] != 0) {
38         for (j = 1; j <= N; ++j) {
39             if (((11 << 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
4 typedef ???      elem_t;          /* a lista_t elemtípusa */
5 typedef struct  cella_t {
6     elem_t adat;                  /* adatelem */
7     struct cella_t *csat;        /* a következő elem */
8 } cella_t;
9 typedef struct  cella_t *pozicio_t;
10 typedef pozicio_t lista_t;
11
12 /* A műveletek deklarációja: */
13 void letesit(lista_t *l);
14 void megszuntet(lista_t *l);
15 void kiolvas(pozicio_t p, elem_t *x);
16 void modosit(pozicio_t p, elem_t x);
17 void beszur(lista_t *l, pozicio_t p, elem_t x);
18 void torol(lista_t *l, pozicio_t p);
19 void kovetkezo(pozicio_t p, pozicio_t *n);
20 void 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 = (cella_t*)malloc(sizeof(cella_t));
29     } else {
30         while (q->csat != p) {
31             q = q->csat;
32         }
33         q->csat = (cella_t*)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 = (cella_t*)malloc(sizeof(cella_t));
65     p->adat = rhs->adat;
66     while ((rhs = rhs->csat)) {
67         p = p->csat = (cella_t*)malloc(sizeof(cella_t));
68         p->adat = rhs->adat;
69     }
70     p->csat = NULL;
71 }
```



Számsorozat legnagyobb közös osztója [1/2]

Inkoszt.c [1–21]

```
1 /* Pozitív egész számok legnagyobb közös osztójának meghatározása.
2  * A hurok ismétléses vezérlés megvalósítása break utasítással.
3  * 1997. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2006. Augusztus 8. Gergely Tamás, gertom@inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 int lnko(int x, int y) {
10     /* x és y legnagyobb közös osztójának meghatározása
11     * Euklidesz algoritmusával.
12     */
13     int m;
14
15     while (y != 0) {
16         m = x % y;
17         x = y;
18         y = m;
19     }
20     return x;
21 }
```

Számsorozat legnagyobb közös osztója [2/2]

Inkoszt.c [23–45]

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

Számsorozat legnagyobb közös osztója₁ [1/2]

Inkoszt1.c [1–21]

```
1 /* Pozitív egész számok legnagyobb közös osztójának meghatározása.
2  * A hurok ismétléses vezérlés megvalósítása ismert szerkezetekkel.
3  * 1997. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2006. Augusztus 8. Gergely Tamás, gertom@inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 int lnko(int x, int y) {
10     /* x és y legnagyobb közös osztójának meghatározása
11     * Euklidesz algoritmusával.
12     */
13     int m;
14
15     while (y != 0) {
16         m = x % y;
17         x = y;
18         y = m;
19     }
20     return x;
21 }
```

Számsorozat legnagyobb közös osztója₁ [2/2]

Inkoszt1.c [23–49]

```
23 int main() {
24     int a, b;
25     int tovább;          /* 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     tovább = !0;
34     while (tovabb) {     /* a hurok ciklus kezdete */
35         printf("?");
36         scanf("%d%*[\n]", &b); getchar();
37
38         if (b == 0) {    /* első kijárat */
39             tovább = 0;
40         } else {
41             a = lnko(a, b);
42             if (a == 1) { /* második kijárat */
43                 tovább = 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]

```
1 /* Másodfokú egyenlet valós gyökeinek meghatározása a
2  * megoldó függvényvel.
3  * 1998. március 31. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2013. augusztus 29. Gergely Tamás, gertom@inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include <math.h>
9 #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: %20.10lf és %20.10lf\n", x, y);
46     } else {
47         printf("Az egyenletnek nincs valószínű 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: %20.10lf és %20.10lf\n", x, y);
53     } else {
54         printf("Az egyenletnek nincs valószínű megoldása!\n");
55     }
56     return 0;
57 }
```



Másodfokú egyenlet gyökei [megoldo függvény változat]

masodfok.c [11–37] helyett

```
1 bool megoldo(double a, double b, double c, double *x1, double *x2) {
2     bool valos = true;                                     /* van-e megoldás */
3     double mx1, mx2;                                     /* munkaváltozók *x1 és *x2 helyett */
4     if (a == 0.0) {
5         if (b == 0.0) {                                   /* az egyenlet elfajuló */
6             valos = false;
7         } else {
8             mx1 = mx2 = -(c / b);
9         }
10    } else {
11        double d;                                        /* a diszkrimináns */
12        d = b * b - 4.0 * a * c;
13        if (d < 0.0) {                                   /* nincs valós gyöke */
14            valos = false;
15        } else {
16            mx1 = (-b + sqrt(d)) / (2.0 * a);
17            mx2 = (-b - sqrt(d)) / (2.0 * a);
18            if (fabs(mx1) > fabs(mx2)) { /* gyökök pontosabb kiszámolása */
19                mx2 = c / (mx1 * a);
20            } else if (mx2 != 0) {
21                mx1 = c / (mx2 * a);
22            }
23        }
24    }
25    *x1 = mx1; *x2 = mx2;
26    return valos;
27 }
```


Minimális C program [1/1]

minimalis.c [1-6]

```
1 /* Minimális C program, ami nem csinál semmit
2  * 1998. Február 26. Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 main() {
6 }
```



Számsorozat jellemzői [1/2]

minimax.c [1–22]

```
1 /* Határozzuk meg egy valós számsorozat legkisebb
2  * és legnagyobb elemét, valamint a sorozat átlagát!
3  * 1997. Október 25. Dévényi Károly, devenyi@inf.u-szeged.hu
4  */
5
6 #include <stdio.h>
7
8 int main() {
9     double vegjel, szam, osszeg, min, max, atlag;
10    int db;                               /* az összegzett elemek száma */
11
12    printf("Ez a program valós számsorozat minimális, \n");
13    printf("maximális elemét és átlagát számolja. \n");
14    printf("Az input sorozatot végjel zárja. \n");
15    printf("Kérem a végjelet! \n");                               /* inicializálás */
16    scanf("%lf", &vegjel);
17    printf("Kérem az input számsorozatot! \n");
18    printf("? \n");
19    scanf("%lf", &szam);
20    min = max = szam;
21    osszeg = 0.0;
22    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
34                                     /* a következő szám beolvasása */
35         printf("?_");
36         scanf("%lf", &szam);
37     }
38                                     /* a ciklus vége */
39     if (db == 0) {
40         printf("Üres számsorozat érkezett.\n");
41     } else {
42         atlag = osszeg / db;
43         printf("Minimum=%10.3lf Maximum=%10.3lf\n", min, max);
44         printf("Az átlag=%10.3lf\n", atlag);
45     }
46     return 0;
```

Sorozat statisztikája

minimax.c (v1.0) [1–26]

```
1 /* Határozzuk meg egy valós számsorozat legkisebb
2  * és legnagyobb elemét, valamint a sorozat átlagát!
3  * 1997. Október 25. Dévényi Károly, devenyi@inf.u-szeged.hu
4  */
5
6 #include <stdio.h>
7
8 int main() {
9     double vegjel, szam, min, max, atlag = 0.0;
10    int db = 0;
11    printf("Kérem a végjelet: ");
12    scanf("%lf", &vegjel);
13    printf("Kérem a sorozatot: \n? ");
14    scanf("%lf", &szam);
15    min = max = szam;
16    while (szam != vegjel) {
17        if (szam < min) {
18            min = szam;
19        } else if (szam > max) {
20            max = szam;
21        }
22        atlag += szam;
23        ++db;
24        printf("? ");
25        scanf("%lf", &szam);
26    }
```

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]

```
1 /* n alatt k értékének kiszámítása
2  * 1997. Október 25. Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int main(int argc, char *argv[]) {
8     int n, k, nak;                /* n alatt k */
9     int i;                        /* ciklusváltozó */
10
11     printf("Kérem n-t és k-t\n");
12     printf("az alatt k értékének kiszámításhoz\n");
13     scanf("%d%d%*[\n]", &n, &k); getchar();    /* beolvasás */
14
15     if (n < k || k < 0) {          /* input adatok jöke-e? */
16         printf("%d alatt %d nem értelmezett!\n", n, k);    /* kiíratás */
17     } else {
18         nak = 1;                  /* inicializálás */
19         for (i = 1; i <= k; ++i) {    /* ciklus */
20             nak = nak * (n - i + 1) / i;
21         }
22         printf("%d alatt %d = %d\n", n, k, nak);    /* kiíratás */
23     }
24     return 0;
25 }
```

A volatile kulcsszó hiánya [1/1]

non-volatile.c [1-26]

```
1 /* A volatile kulcsszó hatásának bemutatása
2  * 2016. December 13. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <pthread.h>
6 #include <stdio.h>
7 #include <unistd.h>
8
9 void *szal(void *arg) {
10     while (scanf("%d", (int*)arg) == 1);
11     *((int*)arg) = 0;
12     printf("Szál_befejezése.\n");
13     return NULL;
14 }
15
16 int main() {
17     pthread_t kiiro_szal;
18     int valtozo = 1;
19     printf("A_program_akkor_fejeződik_be,_ha_egy_0_értékű_input_adatután_egy\n"
20          "egészként_nem_értelmezhető_adatot_adunk_neki.\n");
21     pthread_create(&kiiro_szal, NULL, (void*)&valtozo);
22     while (valtozo);
23     printf("Várunk_a_szál_befejezésére.\n");
24     pthread_join(kiiro_szal, NULL);
25     return 0;
26 }
```

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

P3szog.c [1-25]

```
1 /* n alatt k értékének kiszámítása egy nemrekurzív függvényel és
2 * az értékek elrendezése a Pascal háromszögben.
3 * 1997. Október 31. Dévényi Károly, devenyi@inf.u-szeged.hu
4 * 2013. Augusztus 29. Gergely Tamás, gertom@inf.u-szeged.hu
5 * 2020. Július 30. Jász Judit, jasy@inf.u-szeged.hu
6 */
7
8 #include <stdio.h>
9
10 #define SZAMSZ      5          /* egy szám kiírási szélessége */
11 #define KEPSZ      80         /* a képernyő szélessége */
12
13 int n_alatt_k(int n, int k) {
14     /* n alatt k értékének kiszámítása nemrekurzív függvényel */
15     int i, nak;
16
17     if (n >= k && k >= 0) {          /* input adatok jók-e? */
18         nak = 1;                      /* inicializálás */
19         for (i = 1; i <= k; ++i)      /* ciklus */
20             nak = nak * (n - i + 1) / i;
21     } else {
22         nak = 0;
23     }
24     return nak;
25 }
```


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

P3szog.c [27–45]

```
27 int main() {
28     int n;                               /* a sorok száma */
29     int i, j;                             /* ciklusváltozók */
30
31     printf("Kérem a Pascal háromszög sorainak számát\n");
32     scanf("%d*[\n]", &n); getchar();      /* beolvasás */
33     if (0 <= n && n <= KEPSZ / SZAMSZ - 2) { /* kifér-e a képernyőre? */
34         for (i = 0; i <= n; ++i) {
35             printf("%*c", KEPSZ / 2 - (i + 1) * SZAMSZ / 2 - 1, ' '); /* pozicionálás */
36             for (j = 0; j <= i; ++j) {
37                 printf("%*d", SZAMSZ, n_alatt_k(i, j)); /* kiíratás */
38             }
39             putchar('\n');
40         }
41     } else {                               /* hibaüzenet */
42         printf("%d sor nem fér ki a képernyőre\n", n);
43     }
44     return 0;
45 }
```



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

P3szogr.c [1–22]

```
1 /* n alatt k értékének kiszámítása egy rekurzív függvénnyel és
2 * az értékek elrendezése a Pascal háromszögben.
3 * 1997. Október 31. Dévényi Károly, devenyi@inf.u-szeged.hu
4 * 2015. Október 11. Gergely Tamás, gertom@inf.u-szeged.hu
5 * 2020. Július 30. Jász Judit, jasy@inf.u-szeged.hu
6 */
7
8 #include <stdio.h>
9
10 #define SZAMSZ      5          /* egy szám kiírási szélessége */
11 #define KEPSZ      80         /* a képernyő szélessége */
12
13 int n_alatt_k(int n, int k) {
14     /* n alatt k értékének kiszámítása rekurzív függvénnyel */
15     if (n < k || k < 0) {      /* input adatok jók-e? */
16         return 0;
17     }
18     if (n <= 1 || n == k || k == 0) { /* alapesetek */
19         return 1;
20     }
21     return n_alatt_k(n - 1, k - 1) + n_alatt_k(n - 1, k); /* rek. hívás */
22 }
```

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]

```
1 /* Pénzváltás rögzített árfolyamon.
2  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 int main() {
8     double forint, euro;
9     printf("Hány forintot váltásunk?\n");
10    scanf("%lf", &forint);
11    euro = forint / 384.347;                /* 2023-09-14 */
12    printf("%lf HUF = %lf EUR\n", forint, euro);
13    return 0;
14 }
```



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

penzvalto-fgv.c [1–19]

```
1 /* Pénzváltás rögzített árfolyamon, függvénnnyel megvalósítva.
2  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 #define EURO 384.347 /* 2023-09-14 */
8
9 double valtas(double forint, double arfolyam) {
10     return forint / arfolyam;
11 }
12
13 int main() {
14     double osszeg;
15     printf("Hány forintot váltunk? ");
16     scanf("%lf", &osszeg);
17     printf("%lf HUF = %lf EUR\n", osszeg, valtas(osszeg, EURO));
18     return 0;
19 }
```



Pénzváltás

penzvalto.c (v1.0) [1–16]

```
1 /* Pénzváltás rögzített árfolyamon.
2  *
3  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
4  * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 int main() {
10     double forint, euro;
11     printf("Hány forintot váltssunk?\n");
12     scanf("%lf", &forint);
13     euro = forint / 384.347; /* 2023-09-14 */
14     printf("%lf HUF = %lf EUR\n", forint, euro);
15     return 0;
16 }
```

Pénzváltás

penzvalto.c (v2.0) [1–21]

```
1 /* Pénzváltás rögzített árfolyamon.
2  *
3  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
4  * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 #define EUR 384.347                                /* 2023-09-14 */
10
11 double váltas(double forint, double arfolyam) {
12     return forint / arfolyam;
13 }
14
15 int main() {
16     double osszeg;
17     printf("Hány forintot váltásunk? ");
18     scanf("%lf", &osszeg);
19     printf("%lf HUF = %lf EUR\n", osszeg, váltas(osszeg, EUR));
20     return 0;
21 }
```

Pénzváltás parancssorból

penzvalto.c (v3.0) [1–23]

```
1 /* Pénzváltás rögzített árfolyamon.
2  *
3  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
4  * 2022. Augusztus 30. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include <stdlib.h>
9
10 #define EUR 384.347                               /* 2023-09-14 */
11
12 double valtas(double forint, double arfolyam) {
13     return forint / arfolyam;
14 }
15
16 int main(int argc, char *argv[]) {
17     for (int i = 1; i < argc; ++i) {
18         double osszeg;
19         osszeg = atoi(argv[i]);
20         printf("%.21f HUF = %.21f EUR\n", osszeg, valtas(osszeg, EUR));
21     }
22     return 0;
23 }
```


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 %d\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 }
```

Preprocesszálas [1]

preproc.c [1-20]

```
1 /* Preprocesszálas hatásának bemutatása.
2  * 2009. December 17. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4 #define N 30
5
6 #ifdef DEBUG
7 #define STRING "Debug"
8 #else
9 #define STRING "Release"
10 #endif
11
12 #line 200
13 int main() {
14     int unix;
15     char tomb[N] = STRING;
16     for (unix = N - 1; unix && tomb[unix]; --unix) {
17         tomb[unix] = 0;
18     }
19     return 0;
20 }
```


Prímszámok meghatározása [1/3]

prim.c [1–14]

```
1 /* Határozzuk meg az adott N természetes számnál nem nagyobb
2  * prímszámokat.
3  * 1997. December 6. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2006. Augusztus 15. Gergely Tamás, gertom@inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8
9 #define K (8 * sizeof(kishalmaz_t))
10 #define M 262144LL
11 #define N (M * K)
12
13 typedef long long int kishalmaz_t;
14 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 szítálandó 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 kiírása 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]

```
1 /* Határozzuk meg az adott N természetes számnál nem nagyobb
2  * prímszámokat.
3  * 1997. December 6. Dévényi Károly, devenyi@inf.u-szeged.hu
4  * 2006. Augusztus 16. Gergely Tamás, gertom@inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include "halmaz.h"
9
10 #define N 16777216LL
11
12 int main() {
13     halmaz_t szita;
14     halmaz_elem_t p, s, i;
15     long long int lepes, j;
16
17     szita = halmaz_letesit(N);
18     halmaz_bovit(szita, 2);
19     for (i = 3; i <= N; i += 2) {
20         halmaz_bovit(szita, i);
21     }
22 }
```

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             halmaz_torol(szita, s);
29             s += lepes;
30         }
31         do {                                    /* a következő prím keresése */
32             p += 2;
33         } while ((p < N) && (! halmaz_eleme(szita, p)));
34     }
35
36     j = 0;                                     /* a prímszámok kiírása képernyőre */
37     printf("A prímszámok %lld-ig:\n", N);
38     for (p = 2; p < N; ++p) {
39         if (halmaz_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     halmaz_megszuntet(&szita);
49     return 0;
50 }
```

Prímszámok meghatározása [1/2]

prim-nem-hatekony.c [1-18]

```
1 /* Határozzuk meg az adott N természetes számnál nem nagyobb
2  * prímszámokat.
3  * 2014. Január 24. Gergely Tamás, gertom@inf.u-szeged.hu
4  */
5
6 #include <stdio.h>
7
8 #define N 16777216LL
9
10 int prim_e(long long int n) {
11     long long int d;
12     for (d = 2; d * d <= n; ++d) {
13         if (n % d == 0) {
14             return 0;
15         }
16     }
17     return 1 < n;
18 }
```



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]

```
1 /* Adott n számú embert tartalmazó közösség. Eldöntendő,
2  * hogy riadóláncot alkotnak-e?
3  * 2006. augusztus 9. Gergely Tamás, gertom@inf.u-szeged.hu
4  */
5
6 #include <stdio.h>
7
8 #define N      8                /* a sorozat elemeinek száma */
9 #define ELSO  0                /* az első vizsgálandó elem sorszáma */
10
11 int main () {
12     int lanc[N];                /* a sorozatot tároló tömb */
13     int e, i;                   /* munkaváltozók */
14     int kov;                    /* a következő vizsgálandó */
15     int szam;                   /* számláló */
16
17     printf("Kérem a %d elemű sorozatot, amelyről", N);
18     printf("eldöntöm, hogy riadólánc-e!\n");
```



Riadólánc [2/2]

riadolanc.c [20–46]

```
20     for (i = 0; i < N; ++i) {                                     /* beolvasás */
21         printf("%d. kitértesít", 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];
36         lanc[i] = N;                                           /* továbblépés */
37         ++szam;                                               /* jártam már itt bejegyzése */
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]

```
1 /* Adott n számú embert tartalmazó közösség. Eldöntendő,  
2 * hogy riadoláncot alkotnak-e?  
3 * Ehhez felhasználunk egy függvényt, amely eldönti, hogy  
4 * a kapott sorozat ciklikus permutáció-e?  
5 * 2005. október 13. Gergely Tamás, gertom@inf.u-szeged.hu  
6 * 2014. október 9. Gergely Tamás, gertom@inf.u-szeged.hu  
7 */  
8  
9 #include <stdio.h>  
10 #include <stdbool.h>  
11  
12 #define N      8                                /* a sorozat elemeinek száma */  
13 #define ELSO  0                                /* az első vizsgálandó elem sorszáma */  
14  
15 bool ciklikus(int n, int perm[]) {              /* A paraméter egy tömb */  
16     int kov = ELSO, szam = 0;                  /* következő és számláló */  
17     do {  
18         kov = perm[kov]; szam++;                /* továbblépés */  
19     } while ((kov != ELSO) && (szam != n));  
20     return (kov == ELSO) && (szam == n);  
21 }
```

Ciklikus függvény [2/2]

riadolanc-ciklikus.c [23–47]

```
23 int main () {
24     int lanc[N];           /* a sorozatot tároló tömb */
25     int e, i;             /* munkaváltozók */
26
27     printf("Kérem a %d elemű sorozatot, amelyről", N);
28     printf("eldöntöm, hogy riadolánc-e!\n");
29
30     for (i = 0; i < N; ++i) {           /* beolvasás */
31         printf("%d. kit értesít?", i);
32         scanf("%d%*[\n]", &e); getchar();
33
34         while ((e < 0) || (N <= e)) {
35             printf("Hibás adat!\nKérem újra:");
36             scanf("%d%*[\n]", &e); getchar();
37         }
38         lanc[i] = e;
39     }
40
41     if (ciklikus(N, lanc)) {
42         printf("A számsorozat riadolánc.\n");
43     } else {
44         printf("A számsorozat nem riadolánc.\n");
45     }
46     return 0;
47 }
```

Röppálya határértékeinek számítása [1/1]

roppalya.c [1-26]

```
1 /* Röppálya közelítő számítása.
2  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6 #include <math.h>
7
8 #define G 9.80665
9
10 int main() {
11     double v0, vx, vy, alfa, t, sx, sy;
12     printf("Kezdősebesség(m/s)?\n"); scanf("%lf", &v0);
13     printf("Szög(fok)?\n"); scanf("%lf", &alfa);
14     if (0.0 <= alfa && alfa <= 90.0 && 0.0 <= v0) {
15         vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
16         t = vy / G;
17         sy = G / 2.0 * t * t;
18         printf("A röppálya legnagyobb magasága:\n%lf\n", sy);
19         vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
20         sx = vx * 2.0 * t;
21         printf("A lövedék távolsága földetéréskor:\n%lf\n", sx);
22     } else {
23         printf("Hibás adatok!\n");
24     }
25     return 0;
26 }
```

Röppálya határértékeinek számítása

roppalya.c (v1.0) [1–28]

```
1 /* Röppálya közelítő számítása.
2  *
3  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
4  * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include <math.h>
9
10 #define G 9.80665
11
12 int main() {
13     double v0, vx, vy, alfa, t, sx, sy;
14     printf("Kezdősebesség(m/s)?\n"); scanf("%lf", &v0);
15     printf("Szög(fok)?\n"); scanf("%lf", &alfa);
16     if (0.0 <= alfa && alfa <= 90.0 && 0.0 <= v0) {
17         vy = v0 * sin(alfa / 90.0 * M_PI / 2.0);
18         t = vy / G;
19         sy = G / 2.0 * t * t;
20         printf("A röppálya legnagyobb magasága:\n%lf\n", sy);
21         vx = v0 * cos(alfa / 90.0 * M_PI / 2.0);
22         sx = vx * 2.0 * t;
23         printf("A lövedék távolsága földet éréskor:\n%lf\n", sx);
24     } else {
25         printf("Hibás adatok!\n");
26     }
27     return 0;
28 }
```

Röppálya görbe szimuláció [1/2]

roppalya-struct.c [1–13]

```
1 /* Röppálya közelítő számítása.
2  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6 #include <math.h>
7
8 #define G 9.80665
9
10 typedef struct {
11     double x;
12     double y;
13 } 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)?\n"); scanf("%lf", &v0);
20     printf("Szög(fok)?\n"); scanf("%lf", &alfa);
21     printf("Delta_t(s)?\n"); 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]

```
1 /* Röppálya közelítő számítása.
2  *
3  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
4  * 2018. Augusztus 31. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include <math.h>
9
10 #define G 9.80665
11
12 typedef struct {
13     double x;
14     double y;
15 } 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)?\n"); scanf("%lf", &v0);
22     printf("Szög(fok)?\n"); scanf("%lf", &alfa);
23     printf("Delta_t(s)?\n"); 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]

```
1 /* Síkidomok területének és kerületének kiszámítása.
2  * 2018. Szeptember 19. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <math.h>
6 #include <stdio.h>
7
8 #define PUFFERMERET 128
9
10 enum sikidom_tipus_t {
11     kor, haromszog, teglalap
12 };
13
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 };
```

Síkidomok 1. verzió [2/4]

sikidomok.c [26–54]

```
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 };
34
35 double kor_kerulet(struct kor_tulajdonsagok_t k) {
36     return 2.0 * M_PI * k.r;
37 }
38
39 double kor_terulet(struct kor_tulajdonsagok_t k) {
40     return M_PI * k.r * k.r;
41 }
42
43 double haromszog_kerulet(struct haromszog_tulajdonsagok_t h) {
44     return h.a + h.b + h.c;
45 }
46
47 double haromszog_terulet(struct haromszog_tulajdonsagok_t h) {
48     double s = (h.a + h.b + h.c) / 2.0;
49     return sqrt(s * (s-h.a) * (s-h.b) * (s-h.c));
50 }
51
52 double teglalap_kerulet(struct teglalap_tulajdonsagok_t t) {
53     return 2.0 * (t.a + t.b);
54 }
```

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 terület(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_\u00a0formatumu_\u00a0sor!\n");
98             continue;
99         }
100         printf("T=\u00a0%lf\nK=\u00a0%lf\n", terület(s), kerulet(s));
101     }
102     return 0;
103 }
```

Síkidomok 2. verzió [1/5]

sikidomok-fgv-ptr.c [1–12]

```
1 /* Síkidomok területének és kerületének kiszámítása.
2  * 2018. Szeptember 19. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <math.h>
6 #include <stdio.h>
7
8 #define PUFFERMERET 128
9
10 enum sikidom_típus_t {
11     kor, haromszog, teglalap
12 };
```



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]

```
1 /* A static változót mutatjuk be.
2  * 1997. November 7. Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 #include <stdio.h>
6
7 void stat();
8
9 int main() {
10     int i;                                /* ciklusváltozó */
11     for (i = 0; i < 5; ++i) {
12         stat();
13     }
14     return 0;
15 }
16
17 void stat() {
18     int ideiglenes = 1;                    /* minden hívásnál inicializálódik */
19     static int allando = 1;               /* a program elején egyszer inicializálódik */
20     printf("ideiglenes=%u d allando=%u d\n", ideiglenes, allando);
21     ++ideiglenes;
22     ++allando;
23 }
```

Sztring hosszának meghatározása [1/4]

strlen.c [1–21]

```
1 /* A karaktersorozat hosszának meghatározására függvényműveletet írtunk:
2  * 2009. December 17. Dévényi Károly, devenyi@inf.u-szeged.hu
3  */
4
5 int strlen1(char s[]) {                               /* s hosszának kiszámítása */
6     int i = 0;
7     while (s[i] != '\0') {
8         ++i;
9     }
10    return i;
11}
12
13 /*
14  * Egy másik lehetséges megvalósítás ez volt:
15  */
16
17 int strlen2(char s[]) {                               /* s hosszának kiszámítása */
18     int i;
19     for (i = 0; s[i] != '\0'; ++i);
20    return i;
21}
```

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]

struptr.c [1–24]

```
1 /* A char* és char[] típusok különbsége inicializálás esetén.
2  * A program "pointer" paraméterrel történő indítása futási hibát okoz.
3  * 2006. Augusztus 17. Gergely Tamás, gertom@inf.u-szeged.hu
4  */
5
6 #include <string.h>
7 #include <stdio.h>
8
9 int main(int argc, char *argv[]) {
10     char t[] = "Tömb_vagy_pointer";
11     char *p = "Tömb_vagy_pointer";
12     if (argc == 2) {
13         if (!strcmp(argv[1], "tomb")) {
14             strcpy(t, "Hello!");
15         } else if (!strcmp(argv[1], "pointer")) {
16             strcpy(p, "Hello!");
17         }
18         printf("p: %s\n", p);
19         printf("t: %s\n", t);
20     } else {
21         printf("Használat: \n\t%s_tomb\n\t%s_pointer\n", argv[0], argv[0]);
22     }
23     return 0;
24 }
```


Színusz(x) kiszámítása [1/2]

színusz.c [1-22]

```
1 /* sin(x) közelítő értékének kiszámítása a beépített sin(x)
2 * függvény alkalmazása nélkül.
3 * x értékét transzformáljuk a (-Pi,Pi) intervallumba.
4 * 1997. Október 25. Dévényi Károly, devenyi@inf.u-szeged.hu
5 */
6
7 #include <stdio.h>
8 #include <math.h>
9
10 #define EPSZ      1e-10                /* a közelítés pontossága */
11
12 int main() {
13     double osszeg;                    /* a végtelen sor kezdőösszege */
14     double tag;                       /* a végtelen sor aktuális tagja */
15     double x;                         /* argumentum */
16     double x_orig;                   /* az argumentum értékének megőrzése */
17     double xx;                       /* sqr(x) */
18     int j;                            /* a nevező kiszámításához */
19
20     printf("Kérem sin(x)-hez az argumentumot\n");
21     scanf("%lg%*[^\n]", &x);          /* sor végéig olvasunk */
22     x_orig = x;
```

Színusz(x) kiszámítása [2/2]

színusz.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(%8.5lf)=□%13.10lf\n", x_orig, osszeg);
43 return 0;
44 }
```

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

színusz-dbg.c [1-27]

```
1 /* sin(x) közelítő értékének kiszámítása a beépített sin(x)
2 * függvény alkalmazása nélkül.
3 * 1997. Október 25. Dévényi Károly, devenyi@inf.u-szeged.hu
4 * 2014. Május 19. Gergely Tamás, gertom@inf.u-szeged.hu
5 */
6
7 #include <stdio.h>
8 #include <math.h>
9
10 #define EPSZ      1e-10                /* a közelítés pontossága */
11
12 double színusz(double x) {
13     double osszeg = 0.0;              /* a végtelen sor kezdőösszege */
14     double tag    = x;                /* a végtelen sor aktuális tagja */
15     double xx     = x * x;           /* sqr(x) */
16     int     j     = 2;                /* a nevező kiszámításához */
17
18     do {                               /* ciklus kezdete */
19         osszeg += tag;
20 #if DEBUG > 0
21     fprintf(stderr, "[LOG] Tag=%30.10lf Osszeg=%30.10lf\n", tag, osszeg);
22 #endif
23         tag = -(tag * xx / j / (j + 1)); /* következő tag */
24         j += 2;
25     } while (fabs(tag) >= EPSZ);       /* végfeltétel, ciklus vége */
26     return osszeg;
27 }
```

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

színusz-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(%8.5lf)=%13.10lf\n", x, színusz(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(%8.5lf)=%13.10lf\n", x_orig, színusz(x));
47
48     return 0;
49 }
```

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

szinusz-fgv.c [1-23]

```
1 /* sin(x) közelítő értékének kiszámítása a beépített sin(x)
2 * függvény alkalmazása nélkül.
3 * X értékét transzformáljuk a (-Pi,Pi) intervallumba.
4 * 1997. Október 25. Dévényi Károly, devenyi@inf.u-szeged.hu
5 */
6
7 #include <stdio.h>
8 #include <math.h>
9
10 #define EPSZ      1e-10                                /* a közelítés pontossága */
11
12 double szinusz(double x) {
13     double osszeg = 0.0;                                /* a végtelen sor kezdőösszege */
14     double tag;                                        /* a végtelen sor aktuális tagja */
15     double xx;                                        /* sqr(x) */
16     int    j      = 2;                                /* a nevező kiszámításához */
17
18     while (x < -M_PI) {                                /* transzformálás */
19         x += 2 * M_PI;
20     }
21     while (M_PI < x) {
22         x -= 2 * M_PI;
23     }
```

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

színusz-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(%8.5lf)=%13.10lf\n", x, színusz(x));
43     return 0;
44 }
```

A tér legtávolabbi pontjai [1/3]

ter.c [1-17]

```
1 /* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbt!  
2  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu  
3  */  
4  
5 #include <stdio.h>  
6 #include <stdlib.h>  
7 #include <math.h>  
8  
9 typedef struct {                               /* A tér egy háromdimenziós pontjának típusa */  
10     double x, y, z;  
11 } pont_t;  
12  
13 double tav(pont_t a, pont_t b) {  
14     return sqrt((a.x - b.x) * (a.x - b.x) +  
15                (a.y - b.y) * (a.y - b.y) +  
16                (a.z - b.z) * (a.z - b.z));  
17 }
```



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(*tomb));
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("UUUUUU (%lf, %lf, %lf) és\n", p1.x, p1.y, p1.z);
45     printf("UUUUUU (%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(*tomb));
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("UUUUUU (%lf, %lf, %lf) és\n", P1.x, P1.y, P1.z);
46     printf("UUUUUU (%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]

```
1 /* A háromdimenziós tér pontjai közül keressük meg a két legtávolabbit!
2  *
3  * 2012. Szeptember 5. Gergely Tamás, gertom.inf.u-szeged.hu
4  * 2018. Július 26. Gergely Tamás, gertom.inf.u-szeged.hu
5  */
6
7 #include <stdio.h>
8 #include <stdlib.h>
9 #include <math.h>
10
11 typedef struct {                               /* A tér egy háromdimenziós pontjának típusa */
12     double x, y, z;
13 } pont_t;
14
15 pont_t read_pont() {
16     pont_t p = {0.0, 0.0, 0.0};
17     scanf("%lf_ %lf_ %lf", &p.x, &p.y, &p.z);
18     return p;
19 }
20
21 void write_pont(pont_t p) {
22     printf("(%lf, _%lf, _%lf)", p.x, p.y, p.z);
23 }
24
25 double tav(pont_t p, pont_t q) {
26     return sqrt((p.x - q.x) * (p.x - q.x) +
27                (p.y - q.y) * (p.y - q.y) +
28                (p.z - q.z) * (p.z - q.z));
29 }
```

Tér legtavolabbi 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(*tomb));
49     for (i = 0; i < darab; ++i) {
50         tomb[i] = read_pont();
51     }
52     legtavolabb(darab, tomb, &p1, &p2);
53     printf("A legtavolabbi pontok:\n");
54     printf("uuuuuu"); write_pont(p1); printf("és\n");
55     printf("uuuuuu"); write_pont(p2); putchar('\n');
56     free(tomb);
57     return 0;
58 }
```

A volatile kulcsszó hatása [1/1]

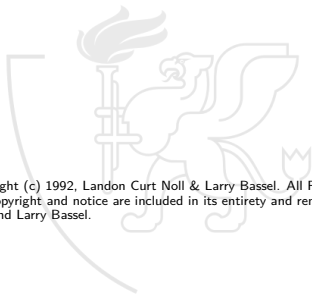
volatile.c [1-26]

```
1 /* A volatile kulcsszó hatásának bemutatása
2  * 2016. December 13. Gergely Tamás, gertom@inf.u-szeged.hu
3  */
4
5 #include <pthread.h>
6 #include <stdio.h>
7 #include <unistd.h>
8
9 void *szal(void *arg) {
10     while (scanf("%d", (int*)arg) == 1);
11     *((int*)arg) = 0;
12     printf("Szál_befejezése.\n");
13     return NULL;
14 }
15
16 int main() {
17     pthread_t kiiro_szal;
18     volatile int változo = 1;
19     printf("A_program_akkor_fejeződik_be,_ha_egy_0_értékű_input_adatután_egy\n"
20          "egészként_nem_értelmezhető_adatot_adunk_neki.\n");
21     pthread_create(&kiiro_szal, NULL, (void*)&változo);
22     while (változo);
23     printf("Várunk_a_szál_befejezésére.\n");
24     pthread_join(kiiro_szal, NULL);
25     return 0;
26 }
```

Hol vagyunk? [1/1]

where.c [1-12]

```
1      main(1
2      ,a,n,d) char**a;{
3      for(d=atoi(a[1])/10*80-
4      atoi(a[2])/5-596;n="@NKA\
5      CLCCGZAAQBEEADAFaISADJABBA^\
6      SNLGAQABDAXIMBAACTBATAHDBAN\
7      ZcEMMCCCCAAhEIJFAEAAAABafHJE\
8      TBdFLDAANEfDNBPHdBcBBBEA_AL\
9      HLELLELLO,uuuuWuOuRLLuD!u"
10     [1++-3];) for(;n-->64;)
11     putchar(!d+++33^
12     l&1);};
```



Copyright (c) 1992, Landon Curt Noll & Larry Bassel. All Rights Reserved. Permission for personal, educational or non-profit use is granted provided that this copyright and notice are included in its entirety and remains unaltered. All other uses must receive prior permission in writing from both Landon Curt Noll and Larry Bassel.