A csoport
Írjon C programot, amely kiszámítja és kiírja a legkisebb olyan négyjegyű számot, melyre igaz, hogy a számjegyeit fordított sorrendben felírva olyan négyjegyű számot kapunk, mely nem egyezik meg az eredetivel, de osztja azt.
Például: a 8712 ilyen, hiszen $8712 = 4 * 2178$.

```c
#include <stdio.h>

int main(void) {
  int number;
  for (number = 1000; number < 10000; ++number) {
    int n0 = number, reverse = 0, j;
    for (j = 0; j < 4; ++j) {
      reverse *= 10;
      reverse += n0 % 10;
      n0 /= 10;
    }
    if (number % reverse == 0 && number != reverse && reverse > 1000) {
      printf("%d\n", number);
      break;
    }
  }
  return 0;
}
```

B csoport
Írjon C programot, amely kiszámítja és kiírja a legnagyobb olyan négyjegyű számot, melyre igaz, hogy a számjegyeit fordított sorrendben felírva olyan négyjegyű számot kapunk, mely nem egyezik meg az eredetivel, de annak egész számú többszöröse.
Például: a 2178 ilyen, hiszen $8712 = 4 * 2178$.

```c
#include <stdio.h>

int main(void) {
  int number;
  for (number = 9999; number >= 1000; --number) {
    int n0 = number, reverse = 0, j;
    for (j = 0; j < 4; ++j) {
      reverse *= 10;
      reverse += n0 % 10;
      n0 /= 10;
    }
    if (reverse % number == 0 && number != reverse) {
      printf("%d\n", number);
      break;
    }
  }
  return 0;
}
```

Ez igen könyű feladatnak bizonyult, sokaknak emiatt lett meg a féléve. A legtöbb megoldás abból indult ki, hogy $\overline{abcd} = 10^3 \cdot a + 10^2 \cdot b + 10 \cdot c + d$. Ekkor a szám és fordítottja négy egymásba ágyazott for ciklussal könnyen generálható. Legközelebb ezért hasonló feladatot 4 helyett a szabványos bemenetről érkező n számjeggyel fogunk feladni.
Tipikus hibák a megoldások során:

- Sokan elmulaszották vizsgálni, hogy a fordított négyjegyű-e.
- Sokan elmulaszották vizsgálni, hogy a fordított megegyezik-e az eredetivel.
- Sokan a fordítást sztringbe írással, számjegycserével és kiolvasással próbálták megoldani. Ez működhet, de figyelni kell a lezáró 0-ra, valamint arra, hogy C-ben nem létezhet olyan, hogy char *s = itoa(n);, csak olyan, hogy char *s = itoa(n, s0); , ahol s0-ban helyet foglaltunk 5 (!) karakternek, és s s0-lal lesz egyenlő.
- Sokan a fordítást számjegyekre bontással végezték: $a = n/1000$, $b = (n - a \cdot 1000)/100$, $c = (n - a \cdot 1000 - b \cdot 100)/10$ stb. . . Ez is működik. Ha viszont ciklusba tesszük, és a 10-hatványokat

a pow() függvénnyel állítjuk elő, akkor figyelnünk kell arra, hogy a pow() double típust ad eredményül, amivel osztva már nem bennfoglalással, hanem valós osztással számolunk. Ekkor persze a fenti algoritmus nem működik. Általában igen nyomós ok kell arra, hogy egy ilyen számelméleti problémába bekeverjük a valós számokat. Ahol a math.h megjelent include-ban, már tudtuk, hogy baj lesz...

- Ha a legkisebb szám megkeresése a feladat, akkor érdemes alulról indulni, és az első találat után leállni. Ha a legnagyobb, akkor felülről indulni, és az első találat után leállni. Mivel megadtunk példában egy lehetséges megoldást, lehetett feltételezni, hogy van megoldás. Így akár n = 1001; while(1) {... n++; } alakban is működött a megoldás. Aki viszont külön energiát fordított arra a lehetőségre, hogy „nincs ilyen szám"...

A-B csoport

Egy programban egész számhalmazokat egyszeresen láncolt strázsa nélküli listákban tárolunk. Definiáljon típust, amely egy ilyen egészeket tartalmazó láncolt listát ír le. Írjon C függvényt, amely paraméterként két listát kap, és visszaadja a két halmaz metszetét/unióját tartalmazó láncolt listát. Az eredmény számára a függvény foglaljon helyet. A halmazokban tárolt elemek sorrendje tetszőleges. Szükség esetén írjon és használjon segédfüggvényt!

```c
#include<stdlib.h>

typedef struct halmazelem {
  int elem;
  struct  halmazelem *kov;
} halmazelem;

halmazelem* uj_halmazelem(int elem) {
  halmazelem *uj = (halmazelem*)malloc(sizeof(halmazelem));
  if(uj != NULL) {
    uj->elem = elem;
    uj->kov=NULL;
  }
  return uj;
}

int eleme(halmazelem *eleje, int elem) {
  halmazelem *akt;
  for(akt=eleje; akt != NULL; akt=akt->kov)
    if(akt->elem == elem)
      return 1;
  return 0;
}

halmazelem *Unio(halmazelem* h1, halmazelem *h2) {
  halmazelem *unio = NULL, *akt;
  for (akt = h1; akt != NULL; akt = akt->kov) {
    halmazelem *uj = uj_halmazelem(akt->elem);
    uj->kov = unio;
    unio = uj;
  }
  for(akt = h2; akt != NULL; akt = akt->kov)
    if(!eleme(unio, akt->elem)) {
      halmazelem *uj = uj_halmazelem(akt->elem);
      uj->kov = unio;
      unio = uj;
    }
  return unio;
}

halmazelem* Metszet(halmazelem *h1, halmazelem *h2) {
  halmazelem *metszet = NULL, *akt;
  for(akt = h1; akt != NULL; akt = akt->kov)
    if(eleme(h2, akt->elem)) {
      halmazelem *uj = uj_halmazelem(akt->elem);
      uj->kov = metszet;
      metszet = uj;
    }
  return metszet;
}
```

A láncolt listákat szénné gyakoroltuk. Alapfeladat. Ne legyenek tipikus hibák!

A-B csoport

Írjon C függvényt, amely két C sztringet kap paraméterként, és azokat szavanként/soronként összefésüli. A sztringekben lévő szavakat/sorokat egyetlen szóköz/újsor karakter választja el egymástól. A két sztringben tárolt szavak/sorok száma nem feltétlenül azonos. Az eredmény számára a függvény foglaljon helyet!

```
#include <stdlib.h> /* malloc */
#include <string.h> /* strlen */

char *interlace(char *s1, char *s2)
{
  char sep = '\n';  /* másik csoport ' ' */
  /* két sztring hossza + szeparátor + lezáró 0 */
  char *res = (char *)malloc((strlen(s1)+strlen(s2)+2)*sizeof(char));
  int i1 = 0, i2 = 0, k = 0, end = 0;
  while (s1[i1] != '\0' || s2[i2] != '\0') {
    while (s1[i1] != '\0' && s1[i1] != sep)
      res[k++] = s1[i1++];
    if (s1[i1] == sep)  /* ha szeparátorhoz értünk, azt is másoljuk */
      res[k++] = s1[i1++];
    else if (!end) {  /* ha a végére értünk, és a másikból még van, szeparátor  */
      res[k++] = sep;
      end = 1;
    }
    /* ugyanaz a másik sztringre */
    while (s2[i2] != '\0' && s2[i2] != sep)
      res[k++] = s2[i2++];
    if (s2[i2] == sep)
      res[k++] = s2[i2++];
    else if (!end) {
      res[k++] = sep;
      end = 1;
    }
  }
  res[k] = '\0';
  return res;
}
```

Tipikus hibák a megoldások során:

- Memóriafoglalás elmarad
- Rossz a szükséges karakterek számának meghatározása.
- A megoldás csak egy összefűzést tartalmaz, nincs meg a külső ciklus
- Sokan megszámolták, hogy melyik sztring a hosszabb, és azzal kezdték az összefűzést, gondolva, hogy akkor abból fog maradni a végére. Ennek semmi értelme. A szavak/sorok különböző hosszúak lehetnek, így a szöveg hossza semmilyen lényegi információt nem tartalmaz.
- A szóközös csoportban sokan felismerték, hogy az sscanf függvény a következő szót kiolvassa a sztringből. Ez így is van, csak nem szabad úgy használni, mint a scanf-et, mert itt nincs a háttérben aktuális fájlpozíció, ami automatikusan lépne a következő szó elejére. Éppen ezért az sscanf(s1, "%s", s); sscanf(s1, "%s", s); sscanf(s1, "%s", s); sorozat mindháromszor az első szót olvassa ki az s1 sztringből és másolja s elejére. (Arról nem is beszélve, hogy aki sscanf(s1, "%s", s0)-lal dolgozik úgy, hogy char s0[32], az . . . nem is mondom.)
- Általában: Ha valaki úgy kezdi a megoldást, hogy odaírja a lapra, hogy „Feltételezem, hogy x legfeljebb y hosszú”, az lényegében azt írja oda, hogy „Erre a feladatra nem kérek pontot”.