# Programozás Alapjai 2. ZH

**13. feladatsor**

Szoftverfejlesztés Tanszék

2022, Ősz

**Feladat** Töltsd le a bíróról a `minta.zip` állományt, majd tömörítsd ki! A `feladat.c` fájlban megtalálod a feladatok megoldás-kezdeményeit. Bővítsd ezt az alább olvasható feladatok alapján! Lehetőség szerint ellenőrizd megoldásod, majd töltsd fel a `feladat.c` fájlt a bíróra!

**Kiértékelés** A bíró lefordítja a programot, majd lefuttatja azt a feladat pontszámának megfelelő számú tesztesettel. Egy teszteset egy bemenet-kimenet pár, amely a megfelelő feladathoz készült. A teszteset akkor helyes, ha az adott bemenethez tartozó kimenet **minden egyes karaktere** megegyezik az előre eltárolt referencia kimenettel. *További feltételek: a program futása nem tarthat tovább 5 másodpercnél, egyszerre nem fogyaszthat többet 16 MiB memóriánál és nem történhet futási hiba (pl. illetéktelen memória hozzáférés).*

**Ellenőrzés** Feltöltés előtt érdemes ellenőrizni a megoldásod.

1. **Fordítás** Ellenőrizd, hogy a programod lefordul-e! A bíró a `gcc -O2 -static -o feladat feladat.c` paranccsal fordít, érdemes ezt használni. A `-Wall` kapcsoló is hasznos lehet.
2. **Példa tesztesetek** Ellenőrizd, hogy a programod helyesen működik-e! A `minta.zip` tartalmaz a bíró által futtatott tesztesetek közül feladatonként egyet-egyet. Az első feladat teszteléséhez másold a programod mellé az `ex1.be` fájlt `be.txt` néven, futtasd le a programod, majd az így kapott `ki.txt` tartalmát hasonlítsd össze az `ex1.ki` fájlban található referencia kimenettel.
3. **Extra tesztesetek** Ellenőrizd a programod működését további példák segítségével! Néhány további teszteset is elérhető, de ezek csupán ellenőrzésre használhatóak, a bíró nem futtatja őket. Ezek használatához futtasd a programod a `-t` vagy `-test` kapcsolóval, például a `./feladat -test` paranccsal. Csak az első feladat teszteléséhez futtasd a programod a `./feladat -t 1` paranccsal.

**1. feladat** (2 pont)
A feladat a siknegyed függvény megírása. A függvény egy síkbeli pont `x` és `y` koordinátáit kapja meg paraméterként, visszatérési értéke pedig a síknegyed száma, amelyben a pont található. (A jobb felső az első, bal felső a második, bal alsó a harmadik, jobb alsó pedig a negyedik síknegyed.) A kapott pont nem illeszkedik egyik tengelyre sem.

Kódold le C nyelven a függvényt! A függvény fejlécén ne változtass! A függvény inputjai a paraméterek, outputja a visszatérési érték. A függvény nem végez IO műveleteket!

```
int siknegyed(double x, double y);
```

**2. feladat** (3 pont)
A feladat meghatározni két egész szám közötti zárt intervallumba eső négyzetszámok darabszámát. A függvény két paramétere sorban az intervallum alsó (`a`) és felső (`b`) végpontja. Visszatérési értéke az intervallumba eső négyzetszámok darabszáma. A végpontok még az intervallum részei.

A feladat a `math.h` használata nélkül megoldható egy ciklussal, amely 0-tól indul, és addig tart amíg a ciklusváltozó négyzete nagyobb nem lesz `b`-nél. A ciklusmagban ellenőrizzük, hogy a ciklusváltozó négyzete nagyobb-egyenlő-e mint `a`, és ha igen, akkor növeljük a négyzetszámok darabszámát tároló változó értékét eggyel. A függvény ezzel az eredetileg 0-ra inicializált változóval tér vissza.

Kódold le C nyelven a függvényt! A függvény fejlécén ne változtass! A függvény inputjai a paraméterek, outputja a visszatérési érték. A függvény nem végez IO műveleteket!

```
int negyzetszam(int a, int b);
```