Rendszerközeli programozás

Gyakorlófeladatok

Az alábbi feladatok megoldásait készítsd el Linux operációs rendszer alatt C programozási nyelven!

1. Írj egy programot, amely kiírja a képernyőre az összes kötőjel (`'-'`) vagy perjel (`'/'`) karakterrel kezdődő parancssori argumentumát (amiket kapcsolóknak is szoktunk hívni)!
2. Írj egy programot, amelyről feltételezzük, hogy egy pozitív egész számnak tekinthető értéket kap parancssori argumentumként és ennek az értéknek négyzetét írja ki a kimenetre!
3. Írj egy programot, amely a saját futtatható állományáról készít egy másolatot `"last"` néven! (Megjegyzendő, hogy a futtatható fájl neve tetszőleges lehet, azaz nem ismert a programírás időpontjában.)
4. Írj egy programot, amely (visszatérési értékként) visszaadja az operációs rendszer számára annak a parancsnak a karakterszámát white space-ek nélkül, ami őt magát elindította! (Például a 12-t a `"./a.out Hello"` parancs esetén.)
5. A `gcc` parancs rengeteg parancssori argumentummal használható, de tekintsük most csak a következő három lehetőséget, amelyek sorrendje tetszőleges.
   - A `gcc`-nek szüksége van egy forrásállomány nevére. Ez egy sztring, ami nem kezdődhet kötőjel (`'-'`) karakterrel. (Ebből generál futtatható állományt.) Ha nincs megadva forrásállomány, akkor hibaüzenetet kapunk.
   - A `"-Wall"` kapcsolóval a fordítás esetleges figyelmeztető üzenetei jeleníthetőek meg. Opcionális kapcsoló.
   - A `"-o"` kapcsoló után megadott karaktersorozat lesz a futtatható fájl neve (nem a forrásállomány neve). Ha nincs ilyen kapcsoló, akkor pedig a fájlnév `"a.out"` lesz. Ha van kapcsoló, de nincs megadva hozzá futtatható név, akkor hibaüzenetet kapunk.

   Írj egy programot, amely a fenti lehetőséget tudja kezelni, bár forráskód fordítás helyett csak egy-egy aktuális tevékenységet leíró szöveget jelenít meg! Példaként a teljesség igénye nélkül néhány lehetséges futtatási parancs, amit a programnak tudnia kell kezelni a fentiek szerint (feltéve, hogy az elkészült program neve `"myprog"`):

```
./myprog
./myprog ab.c
./myprog -Wall ab.c
./myprog ab.c -o run
./myprog -o run ab.c
./myprog -o run ab.c -Wall
./myprog -Wall -o run ab.c
./myprog -o ab.
```

6. Írj egy programot, amely megmondja, hogy az adott felhasználó alapértelmezett könyvtárában vagyunk-e jelenleg! (A `HOME` és a `PWD` környezeti változók segítenek.)
7. Írj egy programot, amely kiír a képernyőre egy `"<user>@<host>"` formátumú szöveget, ahol a `"<user>"` az adott felhasználó nevét, a `"<host>"` pedig az éppen használt számítógép nevét jelenti!
8. Írj egy programot, amely megjeleníti az alapértelmezett hiba felületen az általa elérhető összes környezeti változó nevét!
9. Írj egy egyszerű „Hello World” programot, amely csak Linux operációs rendszer alatt jeleníti meg a képernyőn a szöveget!
10. Írj egy programot, amely a rendszeridő szerint az adott percből hátralévő másodpercek számával tér vissza!

11. Írj egy programot, amely kiírja, hogy hétköznap vagy hétvége van-e a futtatás pillanatában!
12. Írj egy programot, amely megkéri a felhasználót, hogy gépelje be a leghosszabb szót, amit ismer! A program mondja meg, hogy hány másodpercig tartott a gépelés ($\pm$1s tűréshatárral)!
13. Írj egy programot, amely kitalál két valós véletlenszámot a [0 ; 1000] zárt intervallumban, majd ezután kiír a képernyőre egy véletlenszerűen választott számot, amely az előbbi két érték közé esik!
14. Írj egy programot, amely kiír a kimenetre egy maximum 6 számjegyű pozitív egész véletlenszámot a képernyőre! Ha a programot ugyanabban a percben többször is elindítjuk, akkor ugyanazt az értéket írja ki, de ez minden percben egy másik érték legyen!
15. Írj egy programot, amely egy trükkös rulett kereket szimulál, azaz írjon ki egy nullánál nem kisebb és 36-nál nem nagyobb egész számot véletlenszerűen! A 17-es szám 3% eséllyel forduljon elő, az össze többi viszont azonos valószínűséggel szerepeljen!
16. Írj egy programot, amely, amely véletlenszerűen 1000 karaktert ír a kimenetre! Ezek csak az angol ABC kisbetűi, illetve a szóköz karakter lehetnek. A szóköz és a betű karakterek aránya legyen 1:7!
17. A kétdimenziós bolyongás egy olyan folyamat, amely során egy adott időpillanatban azonos valószínűséggel teszünk 1 egység hosszúságú lépést a négy lehetséges irány (fel, le, jobbra, balra) egyike mentén. Majd ezt követően újra és újra megteszünk egy új véletlenszerű lépést. Írj egy programot, amely egy ilyen 2D bolyongást szimulál! A lépések addig ismétlődjenek, amíg a síkon el nem jutunk 100 egység távolságra a kiindulási ponttól. A program végül írja ki az iteráció befejezéséhez szükséges szimulációs lépések számát!
18. Fej vagy írás játékot játszunk alkalmanként 1 zseton tétet alkalmazva. Minden körben a fej dobásra fogadunk. Ha fejet dobunk, akkor nyerünk egy plusz zsetont, ellenben elveszítünk egyet. Kezdetben 10 zsetonunk van. A játékot addig folytatjuk, amíg vagy el nem veszítjük az összes zsetont, vagy amíg meg nem duplázzuk 'vagyonunkat'. A program végül írja ki, hogy hányszor dobtunk a játék végéig!
19. Írj egy programot, amely 1000 darab számjegyet (0-9) ír a képernyőre véletlenszerűen, de nem azonos valószínűséggel! Az egyes számjegyek gyakorisága legyen arányos az értékükkel! Például a 4 kétszer akkora eséllyel forduljon elő, mint a 2; a 3 valószínűsége pedig legyen harmada a 9 esélyének!
20. Írj egy programot, amely skandináv lottó sorsolást szimulál, azaz írass ki 7 darab egész számot 1 és 35 között (beleértve a szélső értékeket is, és figyelembe véve, hogy ismétlés nem lehetséges)!
21. Írj egy programot, amely tartalmazza egy sztringekkel kapcsolatos függvény definícióját! A függvénynek egyetlen paramétere egy sztring címe, visszatérési értéke pedig egy mutató, a paraméterként kapott sztring első nem betű karakterének a címe. (Betű alatt az angol ABC kis és nagybetű karakterei értendőek.) Amennyiben a sztring csak betű karaktert tartalmaz, akkor a függvény `NULL` értékkel térjen vissza! A főprogramban olvass be egy sztringet billentyűzetről, a függvény felhasználásával cserélj le a sztringben minden nem betű karaktert egy szóköz karakterre!
22. Írj egy programot, amely tartalmazza egy kétparaméteres eljárás definícióját, amely két érték felcserélésére szolgál! A két paraméter egy-egy egész típusú változó címe és az eljárásnak az itt található számokat kell felcserélnie. A főprogramban hívd meg ezt az eljárást egy tömb rendezése során!
23. Írj egy programot, amely tartalmazza egy kétparaméteres eljárás definícióját! Az eljárás egy valós számokat tartalmazó tömb értékeinek csökkenő sorrendbe való rendezésére szolgál. (A rendezés algoritmusa általad választott.) Az eljárás első paramétere egy `float` típusú értékeket tartalmazó tömb kezdőcíme, a második a tömb eleminek darabszáma. A főprogramban hozz létre egy tetszőleges méretű és tartalmú tömböt, ezt rendezd az eljárás segítségével, majd írasd ki az elemeket!
24. Írj egy programot, amely tartalmaz egy függvénydefiníciót! A függvény paraméterei két darab egész számokat tartalmazó tömb kezdőcíme és a tömbökben tárolt elemek darabszámai (amelyek eltérőek lehetnek). A függvény foglaljon le egyetlen összefüggő memóriaterületet a két tömb összes eleme számára, ide másolja át az összes számot, és adja vissza a hívónak a terület címét! (Az értékek

sorrendjére nincs kikötés.) A főprogramban hozz létre két különböző méretű tömböt tetszőleges tartalommal, hívd meg rájuk a függvényt és a visszakapott címről írasd ki az összes elemet a képernyőre! A függvényben dinamikusan foglalt memóriaterületet szabadítsd fel a főprogramban annak befejezése előtt!

25. Írj egy programot, amely egy FIFO elven működő sor adatszerkezetet használ, amelyet láncolt lista technikával valósít meg! A programban legyen egy `int` típusú (számlálóként használt) változó, aminek értéke kezdetben 1! A változó pillanatnyi értékét időnként be fogjuk majd tenni a sor végére és ezután mindig megnöveljük a változó értékét eggyel. A program futása során először tedd be a sorba a változó aktuális értékét tízszer egymás után (közben persze legyen inkrementálva), ezután kezdj egy ciklust! Minden iterációs lépésben véletlenszerűen 50% eséllyel vagy írasd ki (és távolítsd el) a sor elején lévő értéket, vagy tedd a sor végére a számláló változó pillanatnyi értéket! Az ciklus akkor fejeződjön be, amikor a sor üressé válik vagy ha a kiírt értékek száma eléri az ezret! A program leállása előtt szabadítsd fel a lefoglalt memóriát!

26. Írj egy programot, amely egyirányban láncolt lista segítségével tárol egész számokat! Az adatokat a láncolás irányában növekvő sorrendben rendezetten kell tárolni megfelelő helyre történő beszúrással. Ments el ide 100 tetszőleges véletlenszámot, majd a láncon végig haladva írasd ki az értékeket szóközzel elválasztva! A program leállása előtt szabadítsd fel a lefoglalt memóriát!

27. Írj egy programot, amely egy kétirányban láncolt lista segítségével tárol karaktereket! (Természetesen a listához tartozni fog egy *fej* és egy *vég* mutató is.) Tárol el a listában a neved betűit! Ezután írasd ki az össze eltárol karaktert először egyik, majd másik irányban végig haladva a láncolásokon! A program leállása előtt szabadítsd fel a lefoglalt memóriát!

28. Írj egy programot, amely bitműveletek segítségével egyetlen `char` típusú változóban tartja nyilván, hogy egy alkalmazott a hét melyik napján/napjain dolgozott! A változó legkisebb helyiértékű bitje jelentse a hétfőt, a második legnagyobb helyiérték pedig a vasárnapot, valamint, ha egy pozícióban `1` bit szerepel, az jelentse azt, hogy az adott napon az alkalmazott dolgozott, a `0` bit pedig, hogy nem! A programban definiálj két `int` típusú függvényt, amelyeknek egyetlen (munkaidő nyilvántartó) `char` típusú paramétere van! Az egyik függvény térjen vissza az adott hét hétvégéjén munkával töltött napok számával, a másik pedig mondja meg, hogy a dolgozó hány hétköznapon nem dolgozott! A főprogramban inicializálj egy karakter változót egy 0 és 127 közötti véletlenszerű értékkel, ezzel hívd meg a függvényeket és írasd ki a főprogramban a visszakapott értékeket!

29. Egy 2 bit színmélységű tömörítetlen kép esetén 2 biten tároljuk egy pixel 4 lehetséges színének indexét (binárisan: 00, 01, 10, 11) sorfolytonosan egy összefüggő memóriaterületen (ez a pixeltömb). Írj egy programot, amelyben először készítesz egy képzeletbeli pixeltömböt, azaz egy 100 bájt méretű memóriaterületet (tömböt) tölts fel 0 és 255 között véletlenszámokat tartalmazó bájtokkal! Ezután bitműveletek segítségével a programnak ki kell írnia, hogy melyik szín(index) hány pixelben jelenik meg! (Például egy 2 bájtos tömb esetén, ha a két bájt értéke 79 és 203, azaz binárisan 01001111 11001011, akkor az egyes színindexek előfordulásai az alábbiak. 00: 2db, 01: 1db, 10: 1db, 11: 4db.)

30. A pakolt BCD számábrázolás esetén egy decimális számjegyet 4 biten tárolunk (a számjegy érték kettes számrendszerbeli alakjának megfelelően) és így két számjegy kerül egy bájtba (szükség esetén vezető nulla biteket használva). Írj egy programot, amelyben egy eljárást hozol létre, amely bitműveletek segítségével egy paraméterként kapott előjel nélküli egész számot BCD kódolású szövegként írat ki a szükséges számú karaktert használva! Hívd meg az eljárást a főprogramban egy tetszőleges pozitív egész számmal! Néhány példa:

paraméter	*BCD kódolt bitsorozat*	*karakterkódok*	*karakterek*
41	01000001	65	"A"
2340	00100011 01000000	35, 64	"#@"
672538	01100111 00100101 00111000	103, 37, 56	"g%8"

31. Az UTF-8 kódolás során egy karakter kétbájtos unicode értékét (`unsigned short int`) átalakítjuk egy bájtsorozattá (felfogható karaktersorozatként is), amely 1, 2, vagy 3 bájtból áll. Az átalakítás az

unicode érték nagyságától függő helyekre előre meghatározott biteket szúr be az alábbi sémák szerint (https://en.wikipedia.org/wiki/UTF-8):

unicode tartomány	*bináris unicode alak*	*bináris UTF-8 alak*
0-127	00000000 0abcdefg	0abcdefg
128-2047	00000abc defghijk	110abcde 10fghijk
2048-65535	abcdefgh ijklmnop	1110abcd 10efghij 10klmnop

Írj egy programot, amelyben egy eljárást hozol létre, amely bitműveletek segítségével egy paraméterként kapott előjel nélküli egész számot (unicode érték) UTF-8 kódolású szövegként írat ki a szükséges számú karaktert használva! Hívd meg az eljárást a főprogramban egy tetszőleges 65536-nál kisebb pozitív számmal! Néhány példa:

paraméter	*unicode bitek*	*UTF-8 kódolt bitsorozat*	`char` *értékek*
66	00000000 01000010	01000010	66
734	00000010 11011110	11001011 10011110	203, 158
22904	01011001 01111000	11100101 10100101 10111000	229, 165, 184

32. Írj egy programot, amelyben bitműveletek segítségével létrehozol egy olyan függvényt, amely a paraméterként kapott `int` típusú érték bájtsorrendjét megfordítja (a bájton belül a bitek sorrendje változatlan maradjon) és az így kapott értékkel tér vissza! Hívd meg az eljárást a főprogramban egy tetszőlegesen választott paraméterrel, majd a visszakapott értékkel ismét és írasd ki a visszakapott értékeket! Például: 308 276 097 → 2 179 686 162 → 308 276 097.

33. Az IPv4-es hálózati címek 32 bites előjel nélküli fixpontos ábrázolású (`unsigned int`) értékek, amelyeket az emberi felhasználó számára pontozott decimális alakban adunk meg. Ez azt jelenti, hogy az érték minden egyes bájtját külön-külön, mint egy-egy 8 bites egész számot íratjuk ki és közéjük pontokat teszünk. Írj egy bitműveleteket használó programot, amely tartalmaz egy eljárást és egy függvényt! Az eljárás paramétere legyen egy előjel nélküli egész szám, amit az eljárás pontozott decimális alakban írat ki a képernyőre! A függvénynek legyen 4 egész szám a paramétere és ezt, mint egy IPv4 címben szereplő számok alakítsa át egyelten `unsigned int` visszatérési értékké! A főprogramban hívd meg a függvényt négy darab 0-255 közötti billentyűzetről beolvasott számmal, a visszakapottértéket pedig ad át paraméterként az eljárásnak!

34. Az IPv4 egy hálózati címek esetén alkalmazott netmaszk egy 32 bites érték, amelynek az elején csak 1-es bitek állnak, ezt pedig csupa 0-s bit követi. A netmaszk prefix hosszának a netmaszkban szereplő 1-es bitek számát nevezzük. Írj egy programot, amely bekér a felhasználótól egy 1-31 közötti egész számot és kiíratja azt az unsigned int típusú értéket, amely egy ilyen prefix hosszúságú netmaszk szerepét tölti be! Például: 21 → 11111111 11111111 11111000 00000000 → 4294965248

35. Egy szöveges fájlban szóközzel elválasztva ismeretlen számú egész érték van elmentve. (Az állomány nevét te adhatod meg.) Írj egy programot, amely minden futás alkalmával a fájl végére írja a fájlban korábban tárolt számok összegét (megtartva a korábbi értékeket is)!

36. Egy szöveges fájlban ismeretlen számú, három számjegyű egész érték van elmentve köztük egy-egy szóközzel. Írj egy programot, amely végighalad a fájl tartalmán és ha 1-es számjeggyel kezdődik valamelyik, akkor azt felülírja a szám duplájával! A többi érték maradjon változatlan!

37. Írj egy programot, amely egy 100 elemű tömbben véletlenszerűen mínusz egymilliónál nagyobb negatív valós számokat tárol el! A programban legyen két alprogram! Egy eljárás, amely paraméterként megkapja az előbbi tömböt és annak tartalmát formázott módon kiíratja egy szöveges fájlba és egy paraméter nélküli függvény, amely az előbbi fájlból beolvassa a 79. számot és ezzel tér vissza. A főprogramban a tömb előállítása után előbb hívd meg az eljárást, majd a függvényt és írasd ki a kapott értéket!

38. Írj egy programot, amely egy 100 elemű tömbben véletlenszerűen mínusz egymilliónál nagyobb negatív valós számokat tárol el! A programban legyen két alprogram! Egy eljárás, amely paraméterként megkapja az előbbi tömböt és annak tartalmát alacsony szintű fájlkezeléssel kiíratja egy bináris fájlba és egy paraméter nélküli függvény, amely az előbbi fájlból beolvassa a 79. számot és ezzel tér vissza. A

főprogramban a tömb előállítása után előbb hívd meg az eljárást, majd a függvényt és írasd ki a kapott értéket!

39. Egy BMP képfájl 43. bájtjával kezdődő 8 bájton a kép szélessége és magassága van eltárolva binárisan 2 darab little-endian bájtsorrendű, előjel nélküli fixpontos számábrázolású, 32 bites értékként. Írj egy programot, amely a parancssori argumentumként megkapja egy BMP képfájl nevét és kiírja a kép felbontásadatait a kimenetre!

40. Egy hagyományos WAV hangfájl 25. bájtjával kezdődő 4bájton a hang Hz egységben mintavételezésének a frekvenciája (sample rate) van eltárolva binárisan egy little-endian bájtsorrendű, előjel nélküli fixpontos számábrázolású, 32 bites értékként. Írj egy programot, amely a parancssori argumentumként megkapja egy WAV hangfájl nevét és kiírja a hang mintavételezési frekvenciáját!

41. Írj egy programot, amelyben összetartozó értékeket szeretnénk egy rekordban (`struct`) tárolni. A rekordnak legyen egy `int`, egy `float`, egy `double`, egy `char` és egy 10 elemű karaktertömb mezője! Ezeket inicializáld az általad választott értékkekkel! Ezt követően a program mentse el a rekord teljes tartalmát alacsony szinten egy bináris fájlba! Írj egy másik programot, amely egy szintén ismeri a fenti rekordszerkezetet és a bináris fájlból visszaolvasva az adatokat, kiírja azokat a kimenetre!

42. Ments le egy legalább 100×100 pixel felbontású BMP képet az internetről! Írj egy programot, amely ennek a képfájlnak az utolsó 1000 bájtját alacsonyszintű fájlkezelés segítségével felülírja 0x55 értékű bájtokkal! Nézd meg a módosított képet!

43. Írj egy programot, amely kiíratja az aktuális munkakönyvtárad `”.txt”` kiterjesztésű elemeinek a neveit szóközzel elválasztva!

44. Írj egy programot, amely tartalmaz egy egész számmal visszatérő függvény, amelynek egy sztring kezdőcíme a paramétere! A függvény térjen vissza 1-gyel, ha a paraméterként megadott sztring egy fájl vagy alkönyvtár neve az aktuális felhasználó alapértelmezett könyvtárában, különben pedig 0 értéket adjon vissza! A főprogramban kérj be egy sztringet a felhasználótól és hívd meg vele a függvényt! A visszatérési értéktől függően írass ki egy `”Létezik”` vagy egy `”Nem létezik”` szöveget!

45. Írj egy programot, amely kiír a képernyőre két számot, az adott könyvtárban lévő fájlok, valamint alkönyvtárak darabszámait!

46. Írj egy programot, amely az aktuális munkakönyvtár szülő könyvtárában lévő fájlok együttes méretét határozza meg és írja ki kilobájt egységekben!

47. Írj egy programot, amely kiírja a standard kimentre az aktuális felhasználó alapértelmezett könyvtárának ”Desktop” nevű alkönyvtárában lévő 1 óránál nem régebben módosított fájlok nevét tabulátorral elválasztva!

48. Írj egy programot, amely az aktuális mappában lévő minden felhasználó számára futtatási joggal rendelkező fájlok nevét listázza ki! (Alkönyvtárak nevét ne jelenítse meg.)

49. Írj egy programot, amely parancssori argumentumként egy számítógép (domain) nevét kapja meg és kiírja a géphez tartozó IP címet a képernyőre!

50. Írj két programot, amely socket programozás segítségével kommunikálni képes egymással! Ha a szerver program kap egy UDP szegmenst a klienstől, ami egy ”Ki van ott?” sztringet tartalmaz, akkor visszaküldi neki az aktuális felhasználó (bejelentkezési) nevét.

51. Írj két programot, amely socket programozás segítségével képes kő-papír-olló játék megvalósítására! Az egyik program egy TCP szerver legyen, amely fogadja két kliens program csatlakozási kérelmét, ezután küld a klienseknek egy játék kezdetére vonatkozó üzenetet. A kliensek a felhasználóiktól kapott „kézjelet” (kő, papír, olló) elküldik a szervernek, aki miután mindkét kliens üzenetét megkapja tájékoztatja őket arról, hogy ki nyert és bontja a kapcsolatot, majd újabb csatlakozásra vár. A kliens, ha meg kapta a szervertől a játék végkimenetelét, akkor tájékoztatja erről a felhasználót, majd leáll.

52. Írj egy programot, amely egy `fork` művelet révén létrehoz egy gyerek folyamatot, majd az is létrehoz egy saját gyerek folyamatot! Az „unoka” folyamat írassa ki a „nagyszülő” folyamatazonosítóját (PID)!

53. Írj egy programot, amely `fork` művelet révén létrehoz két gyerek folyamatot! A fiatalabb gyerek folyamat írassa ki az idősebb gyerek folyamatazonosítóját (PID)!

54. Írj egy programot, amely egy `fork` művelet révén létrehoz egy gyerek folyamatot! Mindkét folyamat írja ki a pozitív egész számok összegét 1-től a saját folyamatazonosítójának (PID) értékéig!

55. Írj egy programot, amely létrehoz egy csővezetéket (pipe) majd egy `fork` művelet révén létrehoz egy gyerek folyamatot! Az gyerek folyamat a saját folyamatazonosítóját a csővezetéken keresztül juttassa el a szülőjéhez, aki hasonlítsa ezt össze a `fork()` függvény által visszaadott értékkel és írassa ki az összehasonlítás eredményét!

56. Írj egy programot, amely egy `fork` művelet révén létrehoz egy gyerek folyamatot! A szülő kérjen be a felhasználótól egy egész számot és mentse el ezt a `”number.txt”` szöveges fájlba! A gyermek feladata az előbbi fájlból a szám beolvasása és kiírása a képernyőre. A két folyamat sikeres működéséhez szinkronizáld őket felhasználói 1-es szignál segítségével!

57. Írj egy programot, amely kitalál két darab maximum 3 számjegyű pozitív egész számot és kiírja őket a képernyőre és kérje meg a felhasználót az összegük kiszámolására! Ezután a program olvasson be egy egész számot a billentyűzetről, de ne hagyjon a felhasználónak 3 másodpercnél több gondolkozási időt! Ha nem kap ennyi időn belül választ, akkor írja ki a `”Lassú!”` sztringet! Felhasználói válasz esetén vizsgálja meg a program, hogy helyes-e az értéke!

58. Írj egy programot, amely végtelen ciklusban egyesével növeli egy `int` típusú változó értékét! Ha a felhasználó `ctrl+c` billentyűkombinációt alkalmaz, a program írja ki mi a változó aktuális értéke és álljon le!

59. A folyamatos futást igénylő szerver alkalmazások esetén gyakran alkalmazzuk a `SIGHUP` szignált arra, hogy leállítás és újraindítás nélkül újra olvastassuk a programmal a (talán azóta megváltozott) konfigurációs fájlját. Írj egy programot amely egy `”conf.txt”` állományból beolvas egy egész számot, majd ezt folyamatosan, másodpercenként egyszer kiírja a kimenetre! Ha a program kap egy `SIGHUP` szignált, akkor olvassa újra a konfigurációs fájlt és annak új tartalmát írja ki ezentúl másodpercenként!

60. Írj egy programot, amely folytonosan egész számokat olvas be billentyűzetről és nyilvántartja ezek összegét! Amikor a program egy megszakítási szignált (`SIGINT`) vagy egy befejeztető szignált (`SIGTERM`) kap, akkor írja ki a számok összegét mielőtt leállna!

61. Egy programban írj egy olyan eljárást, amely hasonlóképpen működik, mint a könyvtári sleep() függvény, azaz a paraméterként megadott ideig a folyamat kerüljön várakozó állapotba, az idő lejárta után viszont folytassa a normál működést! (Használj szignálkezelést és időzítőt!)

62. Írj egy szálkezelést alkalmazó programot, amely bekér egy egész számot a felhasználótól és ezt a számot felhasználva különböző tevékenységeket hajt végre! Egyik tevékenység az adott számnál nem nagyobb pozitív egészek összegét határozza meg. A másik eldönti, hogy az adott érték prímszám-e. A harmadik elmenti az értéket egy szöveges fájlba. A három tevékenységet három (párhuzamosan futó) szál végezze el!

63. Írj egy programot, amely az egymilliónál nem nagyobb prímszámokat keresi meg és íratja ki a képernyőre! A számolás az összes rendelkezésre álló processzormagon fusson párhuzamosan! Figyelj arra, hogy a kis értékekről sokkal könnyebb eldönteni, hogy prímek-e, mint a nagy számokról, ezért lehetőleg a terhelés legyen elosztva a szálak között!

64. Találj ki olyan összetettebb, életszerű problémákat, amelyek megoldásához a félév során használt eszközöt szükségesek! Implementáld a megoldásokat!