Programozás alapjai, gyakorlati anyag

2009-2010 őszi félév

Csernai Kornél

Szegedi Tudományegyetem
Természettudományi és Informatika Kar

2009. december 1.

Tartalomjegyzék I

Tartalomjegyzék II

Tartalomjegyzék III

Tartalomjegyzék IV

Tartalomjegyzék V

Tartalomjegyzék VI

Tartalomjegyzék VII

Elérhetőségek

E-mail

Csernai.Kornel@stud.u-szeged.hu
(csak Stud-os, hivatalos leveleket fogadok)

WWW

`http://www.stud.u-szeged.hu/Csernai.Kornel/`

Fogadóóra

H-6720 Szeged Árpád tér 2.
Demonstrátori szoba (220)
Időpontja: szerda 11-12
(egyeztetés emailben előtte)

Tennivalók

- A kurzus teljesítésének feltételei

Tennivalók

- A kurzus teljesítésének feltételei
- Kabinetes szabályzatok

Tennivalók

- A kurzus teljesítésének feltételei
- Kabinetes szabályzatok
- STUD regisztráció

Tennivalók

- A kurzus teljesítésének feltételei
- Kabinetes szabályzatok
- STUD regisztráció
  - `http://www.stud.u-szeged.hu/`
    STUD-os gond esetén a számítóközpontban levő Help Deskhez kell fordulni: Árpád tér 2. 47. szoba vagy emailben a help@cc.u-szeged.hu címen.

Tennivalók

- A kurzus teljesítésének feltételei
- Kabinetes szabályzatok
- STUD regisztráció
  - `http://www.stud.u-szeged.hu/`
    STUD-os gond esetén a számítóközpontban levő Help Deskhez kell fordulni: Árpád tér 2. 47. szoba vagy emailben a help@cc.u-szeged.hu címen.
- Kabinetes regisztráció

Tennivalók

- A kurzus teljesítésének feltételei
- Kabinetes szabályzatok
- STUD regisztráció
  - `http://www.stud.u-szeged.hu/`
    STUD-os gond esetén a számítóközpontban levő Help Deskhez kell fordulni: Árpád tér 2. 47. szoba vagy emailben a help@cc.u-szeged.hu címen.
- Kabinetes regisztráció
  - `http://www.inf.u-szeged.hu/jelszo`
    Kabinetes gond esetén a rendszergazdákat kell keresni: Irinyi épület 220-as termében vagy emailben a kabinet@inf.u-szeged.hu címen.

Tennivalók

- A kurzus teljesítésének feltételei
- Kabinetes szabályzatok
- STUD regisztráció
  - `http://www.stud.u-szeged.hu/`
    STUD-os gond esetén a számítóközpontban levő Help Deskhez kell fordulni: Árpád tér 2. 47. szoba vagy emailben a help@cc.u-szeged.hu címen.
- Kabinetes regisztráció
  - `http://www.inf.u-szeged.hu/jelszo`
    Kabinetes gond esetén a rendszergazdákat kell keresni: Irinyi épület 220-as termében vagy emailben a kabinet@inf.u-szeged.hu címen.
- A tematika áttekintése

Tennivalók

- A kurzus teljesítésének feltételei
- Kabinetes szabályzatok
- STUD regisztráció
  - `http://www.stud.u-szeged.hu/`
    STUD-os gond esetén a számítóközpontban levő Help Deskhez kell fordulni: Árpád tér 2. 47. szoba vagy emailben a help@cc.u-szeged.hu címen.
- Kabinetes regisztráció
  - `http://www.inf.u-szeged.hu/jelszo`
    Kabinetes gond esetén a rendszergazdákat kell keresni: Irinyi épület 220-as termében vagy emailben a kabinet@inf.u-szeged.hu címen.
- A tematika áttekintése
- Az ETR, STUD, Kabinet rendszerek áttekintése.

Tennivalók

- A kurzus teljesítésének feltételei
- Kabinetes szabályzatok
- STUD regisztráció
  - `http://www.stud.u-szeged.hu/`
    STUD-os gond esetén a számítóközpontban levő Help Deskhez kell fordulni: Árpád tér 2. 47. szoba vagy emailben a help@cc.u-szeged.hu címen.
- Kabinetes regisztráció
  - `http://www.inf.u-szeged.hu/jelszo`
    Kabinetes gond esetén a rendszergazdákat kell keresni: Irinyi épület 220-as termében vagy emailben a kabinet@inf.u-szeged.hu címen.
- A tematika áttekintése
- Az ETR, STUD, Kabinet rendszerek áttekintése.
- A munkakörnyezet megismerése

Linux

- Operációs rendszer; UNIX, System V alapú

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors
  - Biztonságos

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors
  - Biztonságos
  - Megbízható

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors
  - Biztonságos
  - Megbízható
  - Jórészt C-ben íródott
    (ezt a programozási nyelvet használjuk legfőképp ezen a gyakorlaton)

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors
  - Biztonságos
  - Megbízható
  - Jórészt C-ben íródott
    (ezt a programozási nyelvet használjuk legfőképp ezen a gyakorlaton)
  - Multiuser (többfelhasználós)

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors
  - Biztonságos
  - Megbízható
  - Jórészt C-ben íródott
    (ezt a programozási nyelvet használjuk legfőképp ezen a gyakorlaton)
  - Multiuser (többfelhasználós)
  - Multitasking (több processzusos)

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors
  - Biztonságos
  - Megbízható
  - Jórészt C-ben íródott
    (ezt a programozási nyelvet használjuk legfőképp ezen a gyakorlaton)
  - Multiuser (többfelhasználós)
  - Multitasking (több processzusos)
  - Multiplatform
    (x86, MIPS, x86-64, SPARC, DEC Alpha, Itanium, PowerPC, ARM, m68k, PA-RISC, 390, SuperH, M32R, stb...)

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors
  - Biztonságos
  - Megbízható
  - Jórészt C-ben íródott
    (ezt a programozási nyelvet használjuk legfőképp ezen a gyakorlaton)
  - Multiuser (többfelhasználós)
  - Multitasking (több processzusos)
  - Multiplatform
    (x86, MIPS, x86-64, SPARC, DEC Alpha, Itanium, PowerPC, ARM, m68k, PA-RISC, 390, SuperH, M32R, stb...)
- Szabad szoftver

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors
  - Biztonságos
  - Megbízható
  - Jórészt C-ben íródott
    (ezt a programozási nyelvet használjuk legfőképp ezen a gyakorlaton)
  - Multiuser (többfelhasználós)
  - Multitasking (több processzusos)
  - Multiplatform
    (x86, MIPS, x86-64, SPARC, DEC Alpha, Itanium, PowerPC, ARM, m68k, PA-RISC, 390, SuperH, M32R, stb...)
- Szabad szoftver
  - Ingyenesen elérhető

Linux

- Operációs rendszer; UNIX, System V alapú
  - Gyors
  - Biztonságos
  - Megbízható
  - Jórészt C-ben íródott
    (ezt a programozási nyelvet használjuk legfőképp ezen a gyakorlaton)
  - Multiuser (többfelhasználós)
  - Multitasking (több processzusos)
  - Multiplatform
    (x86, MIPS, x86-64, SPARC, DEC Alpha, Itanium, PowerPC, ARM, m68k, PA-RISC, 390, SuperH, M32R, stb...)
- Szabad szoftver
  - Ingyenesen elérhető
  - Könnyen módosítható

Linux (folytatás)

- Különböző disztribúciókban érhető el,
  pl. Ubuntu, Gentoo, Debian GNU/Linux, Fedora Core, Arch, Red Hat, SuSE, UHU.

Linux (folytatás)

- Különböző disztribúciókban érhető el,
  pl. Ubuntu, Gentoo, Debian GNU/Linux, Fedora Core, Arch, Red Hat, SuSE, UHU.
- Elérhető Live CD/DVD formájában is,
  pl. Knoppix, BackTrack, Slax, SystemRescueCD.

Linux (folytatás)

- Különböző disztribúciókban érhető el,
  pl. Ubuntu, Gentoo, Debian GNU/Linux, Fedora Core, Arch, Red Hat, SuSE, UHU.
- Elérhető Live CD/DVD formájában is,
  pl. Knoppix, BackTrack, Slax, SystemRescueCD.
- Virtualizációs lehetőségek, pl.
  VMware Player/Workstation/ESXi, VirtualBox, Virtual PC.

Linux (folytatás)

- Különböző disztribúciókban érhető el,
  pl. Ubuntu, Gentoo, Debian GNU/Linux, Fedora Core, Arch, Red Hat, SuSE, UHU.
- Elérhető Live CD/DVD formájában is,
  pl. Knoppix, BackTrack, Slax, SystemRescueCD.
- Virtualizációs lehetőségek, pl.
  VMware Player/Workstation/ESXi, VirtualBox, Virtual PC.
- Ezen a gyakorlaton Linuxot fogunk használni, mint munkakörnyezet.

Linux (folytatás)

- Különböző disztribúciókban érhető el,
  pl. Ubuntu, Gentoo, Debian GNU/Linux, Fedora Core, Arch, Red Hat, SuSE, UHU.
- Elérhető Live CD/DVD formájában is,
  pl. Knoppix, BackTrack, Slax, SystemRescueCD.
- Virtualizációs lehetőségek, pl.
  VMware Player/Workstation/ESXi, VirtualBox, Virtual PC.
- Ezen a gyakorlaton Linuxot fogunk használni, mint munkakörnyezet.

Linux (folytatás)

- Különböző disztribúciókban érhető el,
  pl. Ubuntu, Gentoo, Debian GNU/Linux, Fedora Core, Arch, Red Hat, SuSE, UHU.
- Elérhető Live CD/DVD formájában is,
  pl. Knoppix, BackTrack, Slax, SystemRescueCD.
- Virtualizációs lehetőségek, pl.
  VMware Player/Workstation/ESXi, VirtualBox, Virtual PC.
- Ezen a gyakorlaton Linuxot fogunk használni, mint munkakörnyezet.

Hasznos jegyzetek

- Rodek Lajos jegyzete
- /pub/progalap/pral-09N-02.ppt

Otthoni gyakorlás

Saját Linux telepítése (javallott)

- A legtöbb Linux disztribúció ingyenesen letölthető, rendelhető. Üres írható lemez ellenében a rendszergazdák is elkészítenek egy példányt.
- Kezdőknek javasolt az Ubuntu (és változatai, pl. Kubuntu, Xubuntu, stb.) használata, mivel igencsak felhasználóbarát.
- Haladóknak kihívást jelenthet pl. egy Gentoo Linux telepítése, amelynek konfigurációja napokba is telhet, de végül sokkal jobban testreszabott, és valamivel gyorsabb rendszert kaphatunk.
- A rendszer telepítéséhez alapvetően szükséges egy üres, használaton kívüli partíció (esetleg még egy a lapozórendszernek). Tipikusan ext2, ext3, ext4, reiserfs, reiser4 típusú filerendszereket készítünk otthoni használatra.

Otthoni gyakorlás (folytatás)

Munkavégzés távoli bejelentkezéssel

- A hallgatók számára elérhető egy, a kabinetes gépekkel megegyező felszereltségű hallgatói szerver: `linux.inf.u-szeged.hu`.
- A munkamenet SSH protokollon(titkosított) keresztül történik, mindenki a kabinetes felhasználói nevét és jelszavát használja.
- Linuxon pl. `ssh hxxxxxx@linux.inf.u-szeged.hu`
- Windowson a PuTTY nevű remek kliens ajánlott.
- Elérhető egy Solaris-os gép is, `solaris.inf.u-szeged.hu`, a bejelentkezés teljesen hasonlóan az előbbihez.

Otthoni gyakorlás (folytatás)

Fájlok átvitele a kabinetes tárhelyről

- A `linux.inf.u-szeged.hu` és a `solaris.inf.u-szeged.hu` gépeken található fájlátviteli szerver (SFTP).
- A bejelentkezés után elérjük a home könyvtárunkat, és a `/pub` könyvtárat is.
- Linuxon pl. `sftp hxxxxxx@linux.inf.u-szeged.hu`
- Windowson a WinSCP nevű kliens ajánlott.

Otthoni gyakorlás (folytatás) I

Linux telepítése virtuális gépen

- A virtuális számítógép egy szoftver, amely szimulálja a számítógép hardverét.
- A virtuális merevlemez tartalmát egy fájlba menti, így nincs szükség külön partícióra.
- A gazda számítógép(pl. Windows) alatt a virtuális számítógép egy programként fut, nincs szükség a számítógép újraindítására, a két munkakörnyezetben egyszerre dolgozhatunk.
- Mindezek mellett egy teljes rendszert kapunk.
- Ajánlott rendelkezni többmagos, különösen VT-x -et, vagy AMD-V -t támogató processzorral.

Otthoni gyakorlás (folytatás) I

Linux telepítése virtuális gépen

- Virtualizációs megoldások pl.:
  VMware Player/Workstation/ESXi, VirtualBox, Virtual PC.
- A VMware Player ingyenesen letölthető, előre elkészített képfájlokkal könnyedén beüzemelhető.

Linkek

- Szegedi Tudományegyetem
- Természettudományi és Informatikai Kar
- Informatikai Tanszékcsoport
- STUD Hallgatói szerver
- Egységes Tanulmányi Rendszer
- Egyetemi könyvtár
- Egyetemi Számítóközpont
- Irinyi Kabinet
- Egyetemi TVSZ, 2008
- TTIK ügyrend
- Egyetemi Számítógépes Infrastruktúra Szabályzat
- NIIF Felhasználói Szabályzat
- Szegedi informatikai gyűjtemény

Házi feladat

1. Keresd meg az SZTE hálózati topológiájának az ábráját az Egyetemi Számítóközpont honlapján!
2. Keresd meg a NIIF/Hungarnet topológiáját az NIIF honlapján!
3. Keresd meg a GÉANT2 topológiáját ábrázoló ábrát a weben!
4. Hányszor (hány félévben) vehető fel egy tantárgy?
5. Maximum hányszor lehet egy félévben ugyanazon tárgyból vizsgázni?
6. Hányszor lehet összesen ugyanazon tárgyból vizsgázni?

Könyvtárszerkezet

Általános tudnivalók

- A fájlrendszer könyvtárakból épül fel. A könyvtárakban fájlok(file-ok) vagy további könyvtárak, esetleg speciális fájlok (szimbolikus link, eszköz fájl, socket fájl, stb.) lehetnek.
- A könyvtárakat a / jel határolja.
- A fájloknak sokszor nincs kiterjesztésük (kivétel pl. `*.conf`, `*.so`, `.c`, stb...), a kiterjesztés egyébként sem határozza meg a tartalmat.

Fontos!

A legtöbb fájlrendszer Linuxon megkülönbözteti a kis- és nagybetűket, így pl. `egYFilE` nem ugyan az, mint `egyfile` így ezek egymás mellett létezhetnek.

Könyvtárszerkezet (Folytatás)

Fontos könyvtárak

- Gyökérkönyvtár: `/`
  A filerendszer legfelső szintű könyvtára.
- Aktuális könyvtár: `.`
  Jelenleg használt könyvtárra hivatkozás.
- Szülő könyvtár: `..`
  Az adott könyvtárt tartalmazó könyvtárra hivatkozás.

Relatív útvonal

Az aktuális könyvtárhoz viszonyított hivatkozás, pl.
`../../../xy/szoveg.txt`, `abc/def/../ghi/`

Abszolút útvonal

A gyökérkönyvtártól(`/`) megadott hivatkozás, pl.
`/tmp/xy/`, `/etc/passwd`

Könyvtárszerkezet (Folytatás)

Jellemző Linuxos könyvtárak

/bin	Futtatható(bináris) állományok
/boot	A rendszerindításhoz(boot) szükséges állományok
/dev	Rendszereszközök könyvtára(merevlemez, optikai lemez, hálózat, stb...)
/etc	Konfigurációs állományok
/home	A felhasználók saját könyvtárai
/root	A `root` felhasználó könyvtára
/tmp	Ideiglenes állományok helye, mindenki által írható

Example

Dokumentáció `man 7 hier`

A felhasználók saját könyvtára a kabinetben

- Minden kabinetes felhasználónak(pl. `h765432`) van egy saját könyvtára: `/home/h765432`.
- A könyvtár tartalmát a hálózaton keresztül érjük el a munkaállomásról, minden munkaállomásról ugyan azt a tartalmat látjuk.
- A home könyvtárak tartalma rendszeresen mentésre kerül (naponta), így egyes adatokat a rendszergazdák vissza tudnak állítani.
- Linux alatt a ~ (tilde) szimbólum egyes környezetekben a home könyvtárt szimbolizálja (a rendszer a jel láttán az aktuális bejelentkezett felhasználó home könyvtárára gondol).
- A `~h765432` a `/home/h765432/`-re utal.

Linux parancsok

Tudnivalók

Tekintsük át a következő jegyzeteket:

- Rodek Lajos jegyzete
- /pub/progalap/pral-09N-02.ppt

Feladatok (mkdir, cd)

Tegyük fel, hogy éppen bejelentkezés után, és egy parancssor(shell) van előttünk.

- mkdir - Könyvtár létrehozása
  1. Készíts egy 'sajat' nevű alkönyvtárat a home könyvtáradban!
  2. Lépj be a 'sajat' alkönyvtárba!
  3. Készíts a 'sajat' alkönyvtárban egy 'masik' alkönyvtárat!
  4. Lépj vissza a home könyvtáradba!
  5. Próbálj meg készíteni egy 'sajat' nevű alkönyvtárat!
  6. Az előző 2 könyvtárat hozd létre egy paranccsal!
- cd - Egy könyvtárba való váltás
  1. Lépj be a saját home könyvtáradba!
  2. Lépj ki a / -be.
  3. Add ki a cd parancsot! Mit csinált?
  4. Lépj be különböző könyvtárakba mind relatív, mind abszolút út használatával!

Feladatok (rmdir, ls)

- rmdir - Egy ÜRES könyvtár törlése
  1. Töröld le a 'sajat' alkönyvtáradat! Miért nem törli?
  2. Töröld le az összes alkönyvtárat a 'sajat' -ban. Majd töröld le a 'sajat'-ot is!
  3. Hogy lehet ezt egyszerűbben?
  4. Mit csinál az `--ignore-fail-on-non-empty` kapcsoló?
- ls - Fájlok listázása
  1. Listázd ki az aktuális könyvtár tartalmát!
  2. Bővebb információkat is szeretnénk látni!
  3. Listázd ki az ÖSSZES fájlt a home könyvtáradban!
  4. Listázd ki az összes alkönyvtár tartalmát!
  5. Nézd meg, hogy milyen jogosultságai vannak egy könyvtárnak!
  6. A fájlméreteknél olvasható méretekkel listázd ki a fájlokat!
  7. A tulajdonosok helyett azok számait (uid, gid) írasd ki!
  8. Alkalmazz rendezést a kilistázáskor!

Feladatok (pwd, cat, tail, head)

- pwd - Aktuális munkakönyvtár
  1. Nézd meg melyik az aktuális munkakönyvtár!
- cat - Fájlok olvasása, tartalom megmutatása, fájl létrehozása
  1. Nézd meg a `/etc/motd` tartalmát!
  2. Nézd meg a `/etc/shadow` tartalmát! Miért nem tudja olvasni?
  3. Listázz ki egy jó nagy fájlt (pl `.bash_history`).
  4. Listázd ki a jó nagy fájlt úgy, hogy számozva legyenek a sorok!
  5. Mit csinál magában a cat parancs?
- tail, head - Programozott olvasás
  1. Írd ki egy fájl első/utolsó 10 sorát!
  2. Egy gyakran változó fájl végét írd ki folyamatosan!

Feladatok (wc, sort, less, more, most)

- wc - Sorok száma
  1. Számold meg hány sor van egy fájlban!
  2. Számold meg hány szó van egy fájlban!
  3. Számold meg hány bájt van egy fájlban!
  4. Nézd meg, mekkora a leghosszabb sor a fájlban!
- sort - Rendezés
  1. Írd ki a `.bash_history` fájl tartalmát rendezve!
- less, more, most - Fájlok olvasása
  1. Nézzd meg a jó nagy fájlt less-el! Mi a különbség a cat-hez képest?
  2. Most nézd meg more-ral. Milyen esetekben jobbak ezek?

Feladatok (cp, mv)

- cp - Fájlok másolása
  1. Másolj át egy fájlt a home könyvtáradban lévő alkönyvtárba!
  2. Másold át mégegyszer!
  3. Változtasd meg a fájl utolsó módosítási idejét!
  4. Updateld az alkönyvtárban lévő fájlt!
  5. Másold át az alkönyvtáradat rekurzívan egy másik alkönyvtárba!
  6. Archiváld az egyik alkönyvtáradat!
  7. Hozz létre egy fájlra szimbólikus linket cp-vel!
  8. Hozz létre egy hardlinket cp-vel a home könyvtáradban lévő fájlra!
- mv - Állományok átnevezése/mozgatása
  1. Nevezd át a home könyvtáradban létrehozott fájlodat!
  2. Mozgasd át a fájlt egy alkönyvtáradba!
  3. Mégegyszer mozgasd át a fájlt egy alkönyvtáradba, de jelezzen a felülíráskor!

Feladatok (rm, ln, du, quota, df)

- rm - Állományok törlése
  1. Töröld le az általad létrehozott fájlokat!
  2. Törölj le egy nem üres könyvtárat!
  3. Törölj le egy nemüres alkönyvtárat rekurzívan!
  4. Alkalmazd a force -t a törlésre!
- ln - Linkek létrehozása
  1. Hozz létre a `/pub/progalap`-ra egy szimbólikus linket!
  2. Hozz létre egy alkönyvtárra egy hard-linket! Mi lett a különbség?
- du, quota, df - Tárterület megállapítása
  1. Nézd meg, mennyi helyet foglalsz a home könyvtáradban!
  2. Csak a végösszeget jelenítsd meg!
  3. Olvasható formában jelenítsd meg az összeget!
  4. Összegezd az egészet!
  5. Nézd meg mennyi a kvótád a home könyvtáradra!
  6. Nézd meg a különböző partíciókon foglalt lemezterületeket!

Feladatok (chmod)

- chmod - Jogosultságkezelés
  1. A home könyvtáradban lévő fájl jogosultságát állítsd 000 -ra!
  2. Adj olvasási jogot a tulajdonosnak!
  3. Adj írási jogot a tulajdonosnak!
  4. Adj futtatási jogot a csoportnak!
  5. Adj írási jogot mindenkineki!
  6. Vond meg mindenkitől az írási jogot!
  7. Egy alkönyvtárban mindennek adj futtatási jogot!
  8. Erről az alkönyvtárról szedd le a futtatási jogot rekurzívan!
  9. Csináld meg most úgy, hogy csak a fájlokról szedje le a futtatási jogot!
  10. Hozz létre egy 000 jogosultságú könyvtárat! Lépj bele! Miért nem lehet belelépni?
  11. Listázd ki a 000 jogosultságú könyvtár tartalmát! Miért ez az eredmény?
  12. Hozz létre egy 600 jogosultságú könyvtárat! Lépj bele! Miért nem lehet belelépni? Mi a különbség az előzőhöz képest?
  13. Listázd ki a 600 jogosultságú könyvtár tartalmát! Miért ez az eredmény?

Feladatok (chown, groups, file, echo)

- chown - Jogosultságkezelés
  1. Változtasd meg egy fájl csoportját!
  2. Változtasd meg a fájl tulajdonosát!
- groups - Csoportazonosítók
  1. Nézd meg milyen csoportokhoz tartozol!
- file - Állomány típusának megállapítása
  1. Másolj át 5 különböző kiterjesztésű fájlt kiterjesztés nélkül a home könyvtáradba!
  2. Nézd meg a különböző fájlok típusát, és nevezd át őket a kiterjesztésükre!
  3. Használd a fájlokat kiterjesztésük szerint! (kép megnézése pl.)
- echo - Kiiratás
  1. Írj ki a képernyőre egy tetszőleges szöveget!
  2. Az újsort ne írd ki a szöveggel együtt!
  3. Szúrj be a szövegbe egy tabulátor karaktert, és írd ki megfelelően a szöveget!

Feladatok (tar, ps, pidof, fg, jobs, kill, killall)

- tar - Állományösszefűzés
  1. Egy alkönyvtár tartalmát fűzd össze egy .tar fájlba!
  2. Fűzz össze több fájlt egy .tar fájlba!
  3. Szedd ki mindkét .tar fájlból a tartalmukat!
  4. Adj hozzá egy fájlt a már létező .tar fájlhoz!
  5. Nézd meg, milyen fájlok vannak a .tar fájlban!
  6. A tar fájl készítésekor egyúttal tömörítsd is bz2 -vel, gzip -el!
  7. A tar fájl kicsomagolásakor a tömörítést is oldd fel!
- ps, pidof, fg, jobs - Processzek kezelése
  1. Indíts háttérben egy processzt!
  2. Nézd meg a pid -jét a pidof paranccsal!
  3. Listázd ki az éppen futó processzeket!
  4. Hozd előtérbe az indított processzt!
- kill, killall - Processzek kilővése
  1. Ölj meg egy processzt! (pid szerint, név szerint)

Feladatok (w, who, finger, last, find, grep, tee, sed)

- w, who, finger - Felhasználói információk
  1. Kérdezd le, kik vannak bejelentkezve a gépre!
  2. Nézd ezt meg a linux.cab-n is!
- last
  1. Nézd meg mikor jelentkeztél be legutóljára!
- find - Állománykeresés
  1. Keresd meg a /etc könyvtárban az összes m-el kezdődő fájlt!
  2. Keresd meg a /etc könyvtárban az összes m-el, vagy b-vel kezdődő fájlt!
- grep, tee - Szűrés
  1. A `.bash_history` fájlban keress rá a következő szövegekre : ls, cd, sajat
  2. A `.bash_history` fájlból nyerd ki azokat a sorokat, melyek nem tartalmazzák az ls mintát!
- sed - Search & Replace
  1. Cseréld ki a `.bash_history` fájlban a 'sajat' mintát valami másra!

Házi feladat I

1. Készíts listát az aktuális könyvtár tartalmáról
2. Készíts listát a home könyvtárad tartalmáról!
3. Készíts részletes listát a home könyvtárad tartalmáról!
4. Készíts részletes és teljes listát a home könyvtárad tartalmáról!
5. Írasd ki a `/pub/ProgramozasAlapjai/eloadas1.html` fájl tulajdonságait a képernyőre!
6. Írasd ki a `/pub/ProgramozasAlapjai/2004` könyvtár tulajdonságait!
7. Írasd ki a `/pub/ProgramozasAlapjai` könyvtárban lévő össze `ppt` kiterjesztésű fájlt!
8. Írasd ki a home könyvtáradban lévő összes rejtett fájlt!
9. Írasd ki az összes rejtett fájl méretét!
10. Írasd ki a `/pub/ProgramozasAlapjai` könyvtárban lévő összes rejtett fájlt!

Házi feladat II

11. Kérj teljes és részletes listát az összes pub könyvtárban lévő 'Prog'-gal kezdődő könyvtár tartalmáról!
12. Hozz létre egy `torlendo` nevű könyvtárat!
13. Hozz létre egy `gyumolcs` nevű könyvtárat! A `gyumolcs` könyvtáron belül hozz létre egy `alma` és egy `korte` nevű könyvtárat is!
14. Mi lesz az `mkdir /gyumolcs/alma/jonatan` parancs eredménye?
15. Mi lesz az `mkdir Adam/Eva` parancs eredménye?
16. Hozz létre egy `auto` nevű könyvtárat és azon belül egy `Audi` nevűt is. Ez egyetlen paranccsal tedd meg.
17. Mi lesz az `mkdir -p Adam/Eva` parancs eredménye?
18. Töröld a `torlendo` nevű könyvtárat!
19. Töröld az `auto` könyvtárban lévő `Audi` könyvtárat egyetlen paranccsal!
20. Mit csinál a `rmdir Adam` parancs?

Házi feladat III

21. Töröld az `Adam` nevű könyvtárat!
22. Töröld a teljes `gyumolcs` könyvtárat!
23. A home könyvtáradban vagy. Mi történik, ha kiadod a `cd ..` parancsot?
24. Mi lesz a `cd ls` parancs eredménye?
25. Mi lesz a `cd .` parancs eredménye?
26. Mi lesz a `cd /` parancs eredménye?
27. Mi lesz a `cd` parancs eredménye?
28. Hozz létre a saját home könyvtáradban egy `szamitogep` nevű könyvtárat, benne egy `billentyuzet` könyvtárat, azon belül pedig egy `ybillentyu` nevűt! Az aktuális könyvtárad legyen a home könyvtárad. Egyetlen utasítással lépj be az `ybillenyu` könyvtárba. Mi a hatása a `cd ../..` utasításnak?

Házi feladat IV

29. Egyetlen utasítással lépj be a `/pub/ProgramozasAlapjai/2007` könyvtárba!
30. A `/pub/ProgramozasAlapjai/2007` könyvtárban vagy, egyetlen utasítással lépj be a home könyvtáradban lévő `szamitogep` könyvtárba!
31. Másold át a `/pub/ProgramozasAlapjai` könyvtárból az `eloadas2.html` fájlt a home könyvtáradba! (A home könyvtáradban vagy!)
32. Másold át a `/pub/ProgramozasAlapjai` könyvtárból az `eloadas3.html` fájlt a home könyvtáradba! (A home könyvtárad `szamitogep` nevű könyvtárában vagy.)

Házi feladat V

33. Készíts egy `progalap` nevű alkönyvtárat a saját home könyvtáradba, és másold bele a `/pub/ProgramozasAlapjai` könyvtárban található össze `.ppt` kiterjesztésű fájt! Készíts egy másolatot a `pral-08N-01.ppt` fájlról `masolat` néven!
34. Másold át a `/pub/ProgramozasAlapjai/2007` könyvtárból a saját home könyvtáradban lévő `progalap` nevű alkönyvtárba a `vezerles.txt` fájlt!
35. Készíts egy másolatot a `vezerles.txt` fájlról `masolat.txt` néven!
36. Mozgasd át a `vezerles.txt` fájlt a home könyvtáradba!
37. Nevezd át a `vezerles.txt` `megtanulando.txt` névre!
38. Töröld a `masolat.txt` fájlt!
39. Mire jó a `cat` parancs? Mutass rá példát!
40. Ments le egy hosszabb fájlt. Nézzük meg oldalanként!

Házi feladat VI

41. Hány sorból áll a `vezerles.txt` fájl?
42. Hány szót tartalmaz egy tetszőleges fájl?
43. Írasd ki egy tetszőleges fájl első 6 sorát!
44. Írasd ki egy tetszőleges fájl utolsó 5 sorát!
45. Írasd ki egy tetszőleges fájl összes olyan sorát, amely 2-es karaktert tartalmaz!
46. Egy könyvtár jogosultsága: `rw-r--r--`. Mit jelent ez?
47. Mit jelent a következő jogosultság: `rwxr-xr-x`?
48. Ki módosíthatja az `r-xr--r--` jogosultságú fájlt?
49. Add meg azt a parancsot, ami egy fájl jogosultságait `rwx--x-w-` -re állítja!
50. Mi a hatása a `chmod 755 vezerlex.txt` utasításnak?
51. Milyen jogosultságot eredményez a 611 utasítás?
52. Állítsd be a `konyv.xml` fájl jogosultságait úgy, hogy senki se írhassa!

Házi feladat VII

53. Mi az eredménye a `chmod = 777 alma.txt` parancsnak?
54. Mi a hatása a következő parancsnak: `chmod 421 korte.txt`?
55. Mi a hatása a következő parancsnak: `chmod go-w alma.txt`?
56. Mi a hatása a következő parancsnak: `chmod go+w alma.txt`?
57. Mi a hatása a következő parancsnak: `chmod a=rw`?
58. Mi a hatása a következő parancsnak: `chmod go=rx`?
59. Mi a hatása a következő parancsnak: `chmod rw=u proba.txt`?
60. A `pelda.txt` fájl jogosultsága a következő: `rwx--x--x`. Mit kell ahhoz tennünk, hogy mindenki futtatni tudja a fájlt?
61. Készíts egy 'sajat' nevű alkönyvtárat a home könyvtáradba!
62. Lépj be ebbe az alkönyvtárba!
63. Készíts egy 'elso' és egy 'masodik' nevű alkönyvtárat!
64. Készíts az 'elso' könyvtarban egy 'utolso' nevű alkönyvtárat!

Házi feladat VIII

65. Készíts egy 'harmadik' nevű, és abban egy 'vegso' nevű alkönyvtárat!
66. Töröld a 'masodik' nevű alkönyvtárat!
67. Töröld az 'elso' könyvtarban az 'utolso' nevű alkönyvtárat!
68. Töröld az 'elso' és 'harmadik' nevű alkönyvtárat!
69. Készíts a home könyvtáradba egy 'sajat2' nevű alkönyvtárat!
70. Lépj be a 'sajat2' nevű alkönyvtárba!
71. Másold át ide a '/pub/ProgramozasAlapjai/2005' könyvtárból a 'vezerles.txt' fájlt!
72. Készíts egy másolatot a 'vezerles.txt' fájlról 'masolat.txt' néven!
73. Mozgasd át a 'vezerles.txt' fájlt a home könyvtáradba!
74. Lépj vissza a home könyvtáradba!
75. Másold be a 'vezerles.txt' fájlt a 'sajat2' alkönyvtárba.
76. Másolj át minden '.txt' végződésű fájlt a 'sajat2' könyvtárból a 'sajat'-ba!

Házi feladat IX

77. Töröld a 'vezerles.txt' fájlt!
78. Töröld a 'sajat2' könyvtárat a teljes tartalmával együtt!
79. Mozgasd az aktuális könyvtárba a 'sajat' könyvtárból a 'vezerles.txt' fájlt!
80. Töröld a 'sajat' könyvtárból a 'masolat.txt' fájlt!
81. Adj meg mindenkinek minden jogot a 'vezerles.txt' fájlra!
82. Vond meg a csoport és az egyéb felhasználók írásjogát a 'vezerles.txt' fájlhoz!
83. Vond meg mindenkitől a futtatás jogát a 'vezerles.txt' fájlhoz!
84. Állítsd be, hogy csak a csoport tudja és csak olvasni a 'vezerles.txt' fájlt!
85. Adj magadnak írás-olvasási jogot a 'vezerles.txt' fájlhoz!
86. Kérj listát az aktuális könyvtár tartalmáról!
87. Kérj teljes listát a kabinetes pub könyvtár tartalmáról!

Házi feladat X

88. Kérj részletes listát a gyökérkönyvtár tartalmáról!
89. Kérj teljes és részletes listát az összes pub könyvtárban lévő 'Prog'-gal kezdődő könyvtár tartalmáról.
90. Kérj listát a könyvtáradban lévő összes rejtett elemről (ne a tartalmukról)!
91. Lépj be a 'sajat' könyvtárba!
92. Kérdezd le, kik vannak bejelentkezve az általad használt gépre!
93. Nézd meg azt is, éppen min dolgoznak!
94. Töröld a 'sajat' könyvtárat, a tartalmával együtt!

BASH

A BASH(Bourne Again SHell) egy nyílt forráskódú héjprogram, széles körben használt. Kiválóan programozható. Mint héjprogram, közvetít a felhasználó és a rendszer között. Parancsokat vár, és feldolgoz.

Eszköz fileok

Az eszköz fileok a `/dev` könyvtárban vannak hagyományosan. Néhány közülük:

/dev/null	Ez az eszköz minden bemenetet elnyel és nem lesz semmi hatása.
/dev/stdin	Szabványos bemenet.
/dev/stdout	Szabványos kimenet.
/dev/stderr	Szabványos hiba csatorna (kimenet).

Átirányítás

- Egy program futása közben három csatornával rendelkezik: bemenet (stdin), kimenet (stdout), hibakimenet (stderr).

Átirányítás

- Egy program futása közben három csatornával rendelkezik: bemenet (stdin), kimenet (stdout), hibakimenet (stderr).
- Ezeket a csatornákat át lehet irányítani külső helyre is, pl. a kimenetet egy fájlba vagy egy másik processz bemenetére.

Átirányítás

- Egy program futása közben három csatornával rendelkezik: bemenet (stdin), kimenet (stdout), hibakimenet (stderr).
- Ezeket a csatornákat át lehet irányítani külső helyre is, pl. a kimenetet egy fájlba vagy egy másik processz bemenetére.
- Az irányításokat a shell balról jobbra értelmezi.

Átirányítás

- Egy program futása közben három csatornával rendelkezik: bemenet (stdin), kimenet (stdout), hibakimenet (stderr).
- Ezeket a csatornákat át lehet irányítani külső helyre is, pl. a kimenetet egy fájlba vagy egy másik processz bemenetére.
- Az irányításokat a shell balról jobbra értelmezi.

Átirányítás

- Egy program futása közben három csatornával rendelkezik: bemenet (stdin), kimenet (stdout), hibakimenet (stderr).
- Ezeket a csatornákat át lehet irányítani külső helyre is, pl. a kimenetet egy fájlba vagy egy másik processz bemenetére.
- Az irányításokat a shell balról jobbra értelmezi.

$< \text{FILE}$ A file beolvasása, átirányítása a standard bemenetre.

$> \text{FILE}$ A standard kimenet file-ba írása (a file felülíródik).

$>> \text{FILE}$ A standard kimenet file-ba írása (a file végére íródik).

$\text{program}_1 \mid \text{program}_2$ program_1 kimenete a program_2 bemenetére kerül.

Átirányítás (példák)

Példák

- `ls | grep 'alma'` — Az `ls` kimenetéből azok a sorok, amelyekben szerepel az `alma` szó.
- `wc < szoveg.txt` — Az `szoveg.txt`-ben található karakterek, szavak, sorok száma.

Egymás utáni parancsok

- Minden parancsnak van egy visszatérési értéke, ez egy egész szám. Egy parancsról azt mondjuk, hogy sikeresen lefutott, ha visszatérési értéke 0.
- `bash`-ben egy sorban több egymás utáni parancsot is kiadhatunk. Ezeket többféleképpen is megtehetjük, aszerint, hogy milyen feltétel mellett szeretnénk, hogy fussanak. Különböző operátorokkal választhatjuk el a parancsokat:

  `&&` A következő parancs csak akkor fut le, ha az előző parancs sikeresen lefutott.

  `||` A következő parancs csak akkor fut le, ha az előző parancs sikertelenül lefutott.

  `;` A következő parancs mindenképp lefut.

Helyettesítő karakterek

Bizonyos speciális helyettesítő karaktereket használhatunk, hogy több, a mintára illeszkedő file-ra is tudjunk hivatkozni egyszerre:

- ? — Egy darab tetszőleges karakterre illeszkedik.
- * — Tetszőleges számú (tehát akár 0) tetszőleges karakterre illeszkedik.
- `[HALMAZ]` — A halmaz elemei közül pontosan egy karakterre illeszkedik. A halmazban megadhatunk kötőjellel(-) elválasztott intervallumokat is.
- A * és ? nem illeszkednek szó eleji `.`-ra.
- Ha egy karaktert nem akarunk speciálisnak tekinteni, akkor azt escape-elni kell, azaz elé egy \ jelet kell rakni.
  Pl. az `a\?b` kifejezés csak az `a?b` kifejezésre illeszkedik és például az `acb`-re nem.

Helyettesítő karakterek (példák)

Példák

Vegyük az `alma`, `ab`, `al`, `bash` kifejezéseket.

- Az `a*` kifejezés illeszkedik az `alma`, `ab`, `al`, kifejezésekre, a többire nem.
- Az `a?` kifejezés illeszkedik az `ab` és `al` kifejezésekre, a többire nem.
- Az `a[a-z]` kifejezés illeszkedik az `ab`, `al` kifejezésekre, a többire nem.

Környezeti változók

- A `bash`-ben léteznek környezeti változók, ezek lényegében szöveges(betű, szám, jel) értékpárok, pl. `HOME=/home/h765432` azt jelenti, hogy a `$HOME` változó értéke legyen `/home/h765432/`.
- A változók értékadásakor a változó nevét csupa nagy betűvel írjuk, $ jelet nem írunk elé. Ekkor, ha létezett már a változó, értéke felülíródik. Üres változónk is lehet, pl. `HOME=`
- A változó értékének lekérdezésekor a változó nevét csupa nagy betűvel írjuk, $ jelet írunk elé.
- Az aktuális változókat a `set` vagy `printenv` parancsokkal tudjuk lekérdezni.
- Egy változót az `unset` paranccsal tudunk megszűntetni.

Fontosabb környezeti változók

- `$PWD` — Aktuális könyvtár
- `$HOME` — Home könyvtár
- `$PS1` — Aktuális prompt (parancssor)
- `$PATH` — A programok kettősponttal elválasztott keresési útvonalai. Amikor nem abszolút hivatkozással adunk meg egy parancsot, a shell ezekben a könyvtárakban (balról jobbra sorrendben) fogja keresni az adott parancsot
  Pl. `/usr/local/bin:/usr/bin:/bin`

Szövegek kezelése

- Ha egy parancs paramétere több szóból áll, idézőjelek közé kell raknuk.
  Pl. `echo "Ez egy tobbszavas parameter"` vagy
  `echo 'Ez egy tobbszavas parameter'`
- A különbséget a " és a ' között az teszi, hogy a " a változókat behelyettesíti, míg a ' nem.
  Pl. `echo "HOME könyvtáram: $HOME"` kimenete
  `HOME könyvtáram: /home/h765432`, míg
  `echo 'HOME könyvtáram: $HOME'` kimenete
  `HOME könyvtáram: $HOME`

Feladatok (Átirányítás, pipe)

- Átirányítás, pipe
  1. Nézd meg a `/etc/motd` tartalmát, és irányítsd át a home könyvtárad egy fájljába!
  2. Másolj össze három fájl tartalmat egy `ossz.txt` fájlba!
  3. Írd ki egy fájl 23-ik sorát!
  4. Számold meg hány fájl van a könyvtárban!
  5. Indítsd el a `yes` programot, a kimenetét irányítsd a `/dev/null` fájlba, majd állítsd meg (stop) a processzt!
  6. Nézd meg mikor jelentkeztél be legutóljára!
  7. A home könyvtáradban lévő összes m-el kezdődő fájltól vond meg az írási jogot! (`find ~/ -name m* -print | xargs chmod -w`)
  8. A messages.txt fájlban keress rá egy tetszőleges mintára, azt mentsd le egy fájlba, és egyszerre jelenítsd is meg! (`tee`)
  9. Cseréld ki a messages.txt fájlban a Firewall mintát valami másra, és az eredményt mentsd el egy fájlban!

Feladatok (SSH, SFTP, wget)

- ssh - Biztonságos távoli parancsvégrehajtás
  1. Jelentkezz be a kabinet linux/solaris szerverére!
  2. Lépj ki a szerverről!
  3. Jelentkezz be a kabinet solaris szerverére úgy, hogy grafikus alkalmazást is indíthass! (`-X`)
- sftp, gftp, scp - Biztonságos fájlátvitel
  1. Létesíts sftp kapcsolatot a kabinet szerverével!
  2. Másold át a `messages.txt` -t, majd vissza!
  3. Listázd ki a távoli könyvtár tartalmát!
  4. Lépj be az távoli gépen a `'sajat'` könyvtárba!
  5. Ellenőrízd a lokális gépen az aktuális könyvtáradat!
  6. Készíts a lokális gépen egy `x` könyvtárat, majd lépj bele!
  7. Hozd le a távoli gépről az összes `'.txt'` végződésű fájlt!
  8. Lépj vissza egy könyvtárat a távoli gépen!
  9. Tedd fel az egyik txt fájlt a távoli gépre!
  10. Szakítsd meg a kapcsolatot!
- wget - Letöltés
  1. Tölts le egy fájlt az internetről, amely elérhető egy URL-n!

Házi feladat I

1. Közvetlenül bejelentkezés után az alábbi parancsok közül melyek írják ki ugyanazt a képernyőre?
   (a) `pwd`
   (b) `echo`
   (c) `echo .`
   (d) `echo ~`
   (e) `echo $PWD`
   (f) `echo $HOME`
   (g) `ls -d`
   (h) `ls -d .`
   (i) `ls -d ~`
   (j) `ls -d $PWD`
   (k) `ls -d $HOME`
   (l) `ls`
   (m) `ls .`
   (n) `ls ~`
   (o) `ls $PWD`

Házi feladat II

(p) `ls $HOME`
(q) `cd`
(r) `cd .`
(s) `cd ~`
(t) `cd $PWD`
(u) `cd $HOME`
(v) `cat`
(w) `cat .`
(x) `cat ~`
(y) `cat $PWD`
(z) `cat $HOME`

2. Ha az alábbi parancsoknál a <C> helyre a `-r` illetve `-R`kapcsolókat írjuk, mi lesz a különbség ugyanazon parancs két lefutása között? (A `dirS` létező könyvtár, `dirD` bejegyzés viszont nem létezik az aktuális könyvtárban.)

(a) `ls <C> dirS`
(b) `cp <C> dirS dirD`

Házi feladat III

(c) `rm <C> dirS`
(d) `chmod <C> dirS`

3. Mi történik ha kiadjuk az alábbi parancsokat?
   (a) `PATH=`
   (b) `HOME=x`
   (c) `PWD=/`
   (d) `PS1='$ '`
4. Mi az eredménye az alábbi parancsoknak? És ha lehagyjuk a végükről a `.` -ot? (Az `x` könyvtár, a `.txt` végű dolgok pedig fájlok.)
   (a) `ls .`
   (b) `cp a.txt .`
   (c) `cp x/x .`
   (d) `cp x/a.txt .`
   (e) `cp *.txt .`
5. Mire jók az alábbi programoknál a felsorolt kapcsolók (# egy számot jelöl)?

Házi feladat IV

(a) `ls: -a -d -l -R -r`
(b) `mkdir: -p -m`
(c) `rmdir: -p`
(d) `mv: -b -f -i -u --reply`
(e) `cp: -b -f -i -l -r -R -s -u`
(f) `rm: -f -i -r -R`
(g) `ln: -s`
(h) `more: -# +#`
(i) `head: -#`
(j) `tail: -# +# -f`
(k) `grep: -A -B -C -e -r -R`
(l) `wc: -c -L -l -m -w`
(m) `du: -a -h -m -s`
(n) `chmod: -R -c`
(o) `ps: -e -f -u`
(p) `kill: -s -9`
(q) `ssh: -X`

Házi feladat V

6. Adott egy fájl. Melyik az (a fájl nevét nem beleszámítva) legrövidebb parancssor, amivel
   (a) minden jogot megvonsz rá?
7. A bejegyzés neve:
   (A) `*`
   (B) `?`
   (C) `-`
   (D) `-f`
   (E) `-r`

   Hogyan tudod:
   (A) Létrehozni fájlként?
   (B) Lemásolni `$HOME` néven?
   (C) Törölni az eredetit?
   (D) Újra létrehozni, de most könyvtárként?
   (E) Belelépni?
   (F) Idemozgatni az előző könyvtárból a `$HOME` fájlt?
   (G) Törölni a fájlt?

Házi feladat VI

(H) Törölni a könyvtárat?

8. Mit csinál az `rm *` parancs, ha az aktuális könyvtárban létezik egy `-r` nevű fájl, és
   (a) ez az egyedüli bejegyzés a könyvtárban?
   (b) csak rejtett fájlok vannak mellette?
   (c) csak fájlok vannak mellette?
   (d) csak könyvtárak vannak mellette?

A C programozási nyelv

- A nyelv szintaxisa viszonylag kicsi.
- Főbb felhasználási területei: operációs rendszerek, hardverek programozása
  (alacsony szintű programozásra is alkalmas)
- Hatékony fordítók léteznek (pl. GCC optimalizációi)
- Rengeteg platformra létezik fordító
- A nyelv nem rendelkezik file kezeléssel, matematikai függvényekkel; ezeket külön könyvtárakból kell betölteni.
- Szabványos fájl típusok:

.c	C source (forrás) fájl
.h	C header (fejléc) fájl
.i	C preprocessed (preprocesszált) fájl
.s	assembly (gépi) nyelvű fájl
.o	object (tárgykódú) fájl
a.out	link edited output (összeszerkesztett futtatható fájl)

Programozás C nyelven I

- Legfőbb fordítóprogramok:
  - UNIX-ra: GCC (*GNU Compiler Collection*, korábban *GNU C Compiler*) A GCC-nek vannak kiegészítései a C nyelvre nézve, ezek persze nem szabványosak, de segítik a programozót.
  - Windows-ra: MSVC, illetve GCC a Cygwin nevű környezetben
  - Intel C/C++ Compiler, fizetős szoftver (Linux, Windows)
- Fejlesztői rendszerek (IDE-k, *Integrated development environment*-ek):
  - Anjuta (Linux)
  - Dev-C++ (Windows, de Linuxon sem lehetetlen futtatni)
  - NetBeans (Linux, Windows)
  - Fejlesztői környezetek összehasonlítása (Wikipedia): `http://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments#C.2FC.2B.2B`
- Egy másik megoldás, ha sima szövegszerkesztővel elkészítjük a C programunkat, majd kiadjuk a `gcc` parancsot.

Programozás C nyelven II

- Szövegszerkesztők Linuxon:
  - Konzolos:
    - mcedit
    - nano
    - vi, vim
    - emacs
  - Grafikus:
    - kedit
    - kate
    - gedit
- Szövegszerkesztők Windowson:
  - Notepad++
- Szövegszerkesztők összehasonlítása (Wikipedia): `http://en.wikipedia.org/wiki/Comparison_of_text_editors`

A C szintaxisával kapcsolatban néhány gondolat I

- A nyelv érzékeny a kis- és nagybetűkre.
- A nyelv funkcionális, a programunkat függvényekkel kell (érdemes) megírnunk.
- Egy függvénynek lehet bemenete (paraméterek) és kimenete (visszatérési érték), de egyik sem kötelező.
- A függvény paramétereit a függvény neve után zárójelbe tesszük, vesszővel elválasztva felsoroljuk.
- Minden utasítás után pontosvesszőt teszünk.
- Érdemes indentálni a kódot, hogy átlátható legyen (az egyes blokkokat beljebb tolva írni), az üres karakterekből (újsor, szóköz, tabulátor) bármennyit felhalmozhatunk a kifejezések között.
- A fájl végén újsor karakter legyen.
- UNIX alatt az újsor karakter `\n`, Windows alatt két karakter: `\r\n`.

A C szintaxisával kapcsolatban néhány gondolat II

- A fájlokban érdemes kommenteket elhelyezni, hogy a kód jobban érthető legyen, akár később is. A kommenteket a fordító figyelmen kívül hagyja, a preprocessing alatt elhagyja.
  A kommenteket a /* és */ közé kell helyezni. A C99 szabvány bevezette a // kezdetű kommentet, amely a sor végéig tart.
- A változó nevek a következő karakterekből állhatnak:
  - angol ábécé kis és nagy betűi.
  - számjegyek (nem kezdődhet vele)
  - _
- A változónevek nem lehetnek fenntartott szavak.
- Ha ékezeteket használunk (kommentek, sztringek, stb.), akkor a file lehetőleg legyen *UTF-8* kódolású.

C programok fordítása GCC-vel

- Tegyük fel, hogy a `program.c` fájlban elkészítettük a C programunkat.
- Ekkor a `gcc program.c` paranccsal tudjuk lefordítani a programot. Ekkor a kész futtatható tárgyfájl `a.out` néven fog létrejönni, amelyet futtathatunk a `./a.out` paranccsal.
- Amennyiben más néven szeretnénk a futtatható állományt létrehozni, használjuk a `-o` kapcsolót: `gcc -o program program.c`, majd `./program`
- Ha egy C file kiterjesztése `.c`, pl. `program.c`, akkor a `make program` (nincs `.c` a parancs végén) paranccsal is lefordíthatjuk a programot (ez olyan, mintha `gcc -o program program.c`-t írnánk).

Első C program

Az itt bemutatott programok nagy része megtalálható a `/pub/ProgramozasAlapjai/Gyakorlat/gyak04` könyvtárban. Ott vannak további feladatok, akár házi feladat jelleggel is, érdemes foglalkozni velük. A forrás fájlok egy része a honlapomon is fent van.

Írj egy programot, amely nem csinál semmit! (minimal.c)

```
main() {
}
```

Fordítsuk le a programot!

```
gcc -o minimal minimal.c
```

Futtassuk le a programot!

```
./minimal
```

Szöveg kiírása

Írj egy programot, amely kiír a képernyőre egy szöveget! (helloworld.c)

```
#include <stdio.h>

main() {
    printf("Hello Világ!\n");
}
```

Fordítsuk és futtassuk le a programot!

```
gcc -o helloworld helloworld.c && ./helloworld
```

A main() függvény visszatérési típusának int-nek kellene lennie, és kell egy újsor a file végére, javítsuk ki ezeket (a visszatérési érték legyen 0).

Szöveg kiírása, javítva

Javított változat (helloworld0.c)

```
#include <stdio.h>

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

Fordítsuk és futtassuk le a programot!

```
gcc -o helloworld0 helloworld0.c && ./helloworld0
```

Szöveg kiírása, 1 visszatérési értékkel

Jelezzünk az operációs rendszer felé hibát! (helloworld1.c)

```
#include <stdio.h>

int main() {
    printf("Hello Világ!\n");
    return 1;
}
```

Fordítsuk le a programot!

```
gcc -o helloworld1 helloworld1.c && ./helloworld1
```

Visszatérési értékek

Nézzük meg, hogyan működik a visszatérési érték!

```
./helloworld0 ; echo $?
./helloworld1 ; echo $?
./helloworld0 && ./helloworld1
./helloworld1 && ./helloworld0
./helloworld0 || ./helloworld1
./helloworld1 || ./helloworld0
./helloworld0 ; ./helloworld1
```

C nyelvi figyelmeztetések, hibák

- Figyelmeztetés (warning): a programban valószínűleg nem azt írtuk, amit szerettünk volna, illetve kisebb logikai hiba van. A programunkat le tudjuk fordítani, és lehet, hogy hibátlanul fut.
- Hiba (error): a programban szintaktikai, vagy más egyéb súlyos hiba van. A programunkat nem is tudjuk lefordítani.
- Ha a programunk nem fordult le, érdemes az első hibával (error) foglalkozni.
- Minden hibajelzéshez a fordító megadja, hogy hanyadik sor (esetleg hanyadik karakter) pozíción van a probléma. Érdemes ott kutatni, de az is előfordulhat, hogy a forrás fájl javításához nem ott kell módosítanuk, hanem pl. előtte.

C nyelvi figyelmeztetések, hibák

Hasznos GCC kapcsoló

A GCC `-Wall` kapcsolója nagyon hasznos, főleg kezdő programozóknak, mert kiszűri a naív hibákat
pl. nincs változó inicializálva, nincs változó használva, stb.
Ezt mindig javasolt használni.

Példa GCC hibaüzenet

```
x.c:3: error: syntax error before '}' token
```

Magyarázat: Az x.c fájl 3. sorában egy súlyos hiba van, a `}` jel előtt (jelen esetben hiányzott a pontosvessző az utasítás végéről).

A továbbiakban egy `program.c`-t a `gcc -o program -Wall program.c` paranccsal fordítjuk le és a `./program` paranccsal futtatjuk.

C nyelvi kifejezések

Egy matematikai művelet (kifejezes.c)

```
main() {
  3 + 5
}
```

Látjuk, hogy ez a program hibás. Próbáljuk meg lefordítani, majd javítsuk ki!

Egy matematikai művelet, javítva (kifejezes.c)

```
int main() {
    3 + 5; /* Egészítsük ki pontosvesszövel */
    return 0;
}
```

C nyelvi kifejezések

Feladat: Egészítsd ki az előző programot, csinálj utasítást a következő értékekből/értékeketet kiszámító kifejezésekből:

- Egy év napjainak száma.
- Mikor született az, aki most 18 éves?
- Átlagban hány órát kell hetente otthon a progalap gyakorlásával tölteni a szorgalmi időszakban, ha egy kredit (a teljes félév során elvégzett) 30 munkaórát jelent, a félév 20 hétből áll, és ebbe a kreditszámba az órai munka is beleszámít?

C nyelvi kifejezések (megoldás)

Megoldás (kifejezes.c)

```
int main() {
    3 + 5; /* Egészítsük ki még egy pár utasítással */
    365;
    2007 - 18;
    (10 * 30) / 20 - (4 + 3);
    return 0;
}
```

A program ugyan egymás után (szekvenciálisan) elvégzi a műveleteket, de az eredményét nem fogja sehol sem felhasználni (ezt jelzi is `-Wall`).
Házi feladat: egészítsük ki a programot úgy, hogy kiszámítsa és kiírja az eredményeket!

C változók

Feladat: Deklarálj egy valós, egy karakter és két egész típusú változót!

Változók (osszevont.c)

```
int main() {
  float valos;
  char karakter;
  int egesz1, egesz2; /* Több változó ugyan olyan
típusú lehet, ezek deklarációját csoportosíthatjuk. */
  return 0;
}
```

C változók, értékadás

Feladat: Inicializáld a valós értéket `3.14` -re, a karaktert a nagy `A` karakterre, és művelettel adj értéket a két egész változónak is.

Változók értékadással (osszevont.c)

```
int main() {
  float valos = 3.14; /* A pont a határoló karakter */
  char karakter = 'A'; /* Egy darab karaktert
a ' jelek közé rakunk */
  int egesz1, egesz2;
  egesz1 = 3; /* Az értékadás jele az = és ez egy
müvelet */
  egesz2 = 5;
  return 0;
}
```

C deklarációs hiba

Feladat: Próbáld ki, mi történik, ha két deklaráció közé mondjuk egy értékadás műveletet szúrsz!

Változó értékadás hiba (deklhiba.c)

```
int main() {
  float valos;
  valos = 3.14;
  char karakter;
  karakter = 'A';
  int egesz1=3, egesz=5;
  return 0;
}
```

C-ben a deklaráció a blokk elején kell, hogy legyen! Meg kell különböztetni az inicializálást az értékadástól! Az inicializálás a deklaráció (elhagyható) része, az értékadás viszont már művelet!

C deklarációs hiba I

Feladat: Írj egy programot, amelyben van plusz két blokk. Mindegyik deklaráljon egy-egy saját változót. Próbáld ki, hol tudsz a programban ezekre hivatkozni! (blokkhiba.c)

```
int main() {
  int elso;
  elso = 3;
  {
    int masodik;
    elso    = 6
    masodik = 5;
  }
  {
    int harmadik;
    elso    = 9
    masodik = 10;
```

C deklarációs hiba II

```
    harmadik = 8;
  }
  masodik = 15;
  harmadik = 16;
  return 0;
}
```

Feladat: Nézzük meg, mely sorokban voltak hibák! Javítsuk ezeket!

C input/output

Az input/output függvénykönyvtár használatához be kell töltenünk az `stdio.h`-t, a fájl elején lehetőleg:

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

Linuxon az összes fejléc fájl az `/usr/include` alatt van. A gcc-nek további fejléc könyvtárakat adhatunk meg a -I kapcsolóval. Pl. ha azt szeretnénk tudni, hogy az `fscanf` függvény melyik fejléc fájlban található (mert mondjuk éppen ezt szeretnénk include-olni), akkor kiadhatjuk a `grep fscanf /usr/include/ -R -w` parancsot.

C kiíratás

Feladat: Írasd ki az 'X' karaktert, a 2007 egész és a 3.1415 valós számokat, illetve a "Szöveg kiíratása" sztringet a standard kimenetre, mindegyiket új sorba!

Értékek kiíratása (kiiratas.c)

```
#include <stdio.h>

int main() {
  printf("Szöveg kiíratása\n");
  printf("%c\n", 'X');
  printf("%d\n", 2007);
  printf("%f\n", 3.1415);
  printf("%s", "Szöveg másként\n");
  return 0;
}
```

C beolvasás, kiíratás I

Feladat: Olvass be egy egész, egy valós és egy karakter értéket a standard bemenetről, majd írasd ki őket a standard kimenetre! (beolvasas.c)

```
#include <stdio.h>

int main() {
  int    egesz;
  float  valos;
  char   karakter;
  printf("Kérek egy egész számot: ");
  scanf("%d", &egesz);
  printf("Kérek egy valós számot: ");
  scanf("%f", &valos);
  printf("Kérek egy karaktert: ");
  scanf("%c", &karakter);
  printf("Az eltárolt egész szám: %d\n",          egesz);
```

C beolvasás, kiíratás II

```
  printf("Az eltárolt valós szám: %f\n",          valos);
  printf("Az eltárolt karakter: %c\n",            karakter);
  return 0;
}
```

- Fontos, hogy ha beolvasunk, akkor a & jel ott legyen a változó neve előtt, mert a scanf egy memóriacímet vár, és különben a változó értékét venné memóriacímnek, nem pedig a változó memóriacímét. Kivétel: `char*`, azaz sztring típus, mert arra eleve memóriacímmel (pontosabban pointerrel, azaz mutatóval) hivatkozunk.
- Érdemes a megfelelő típusú értékekhez a megfelelő %-os formátumot használni. Persze ennek ellenére is működhet a programunk, és néha van is értelme (pl. ha egy karaktert a decimális ASCII kódja alapján akarunk bekérni).

C beolvasás, kiíratás III

- Fontos, hogy a olyan típust ne adjunk meg formátumnál, amely a hivatkozott változó méreténél (sizeof) nagyobb, mert akkor a változó tárterülete utáni helyre is írni fog a scanf, amely legtöbbször nem az, amit szeretnénk (ezt egyébként -Wall jelzi is).

C kiíratás I

Feladat: Deklarálj és inicializálj egy egész, egy valós és egy karakter változót. Írasd ki mindhárom értékét egészként, valósként és karakterként is! Figyeld meg az eredményt! (fontos.c)

```
#include <stdio.h>

int main() {
  int    egesz = 13;
  float  valos = 0.12345678901234567890;
  char   karakter = 'A';

  printf("Egész egészként kiírva: %d\n", egesz);
  printf("Valós egészként kiírva: %d\n", valos);
  printf("Karakter egészként kiírva: %d\n", karakter);

  printf("Egész valósként kiírva: %f\n", egesz);
```

C kiíratás II

```
  printf("Valós valósként kiírva: %f\n", valos);
  printf("Karakter valósként kiírva: %f\n", karakter);

  printf("Egész karakterként kiírva: %c\n", egesz);
  printf("Valós karakterként kiírva: %c\n", valos);
  printf("Karakter karakterként kiírva: %c\n", karakter);
  return 0;
}
```

Figyeljük meg, hogy a gcc felismeri, ha a típus nem megfelelő, de ez számára nem hiba, csak figyelmeztetés! Valósként kiírva pedig meglepő lesz, hogy a megjelenő érték független az aktuális paramétertől.

C kiíratás, hiba

Feladat: Olvastass be egy double típusú értéket (%lf) egy karakter változóba, és futtasd a programot.

Hibás beolvasás (hiba.c)

```
#include <stdio.h>

int main() {
  char c;
  printf("Írd be: 12345.6789\n");
  scanf("%lf", &c);
    /* Azért %lf, hogy biztosabb legyen a segfault
(de így sem 100%) */
  printf("Sajnos nem volt hiba\n");
  return 0;
}
```

C többszörös beolvasás/kiíratás I

Feladat: Olvass be közvetlenül egymás után egy karakter, egy egész és még egy karakter értéket! A használt változókat inicializáld, hogy láthasd az eredményt! Próbálj ki többféle inputot, és nézd meg, a beolvasás miket talált! (tobb.c)

```
#include <stdio.h>
int main() {
  int    egesz = 0;
  char   k1 = 'X', k2 = 'Y';

  printf("Beolvasás (karakter egész karakter): ");
  scanf("%c%d%c", &k1, &egesz, &k2);
  printf("egesz == %d; k1 == '%c';k2 == '%c';\n",
    egesz, k1, k2);
  return 0;
}
```

C többszörös beolvasás/kiíratás II

Próbáljuk meg, mi történik, ha a következő inputokat adjuk:

- 123
- 123ab
- ab123

C feladatok I

Írd meg az alábbi programokat C nyelven és

(A) készíts belőlük futtatható programot parancssorból egy lépésben!

(B) fordíts belőlük object fájlokat, majd ezekből készíts futtatható programot!

(C) készíts belőlük futtatható programot az anjuta segítségével!

A programokra a gcc fordító ne jelezzen warning-okat -Wall kapcsoló esetén sem!

Írj egy programot, ami

1. kiírja, hogy `Helló Világ!` !
2. kiírja egy általad választott vers első versszakát!
3. kiírja egy általad választott vers első négy versszakát, a versszakokat egy-egy üres sorral elválasztva!
4. bekér egy egész számot, majd kiírja azt!
5. bekér egy valós számot, majd kiírja azt!

C feladatok II

6. bekér két egész számot, majd kiírja az összegüket!
7. bekér két egész számot, majd kiírja a különbségüket (elsőből a második)!
8. bekér két egész számot, majd kiírja a szorzatukat!
9. bekér két egész számot, majd kiírja az egészosztás szerinti hányadosukat (első per második)!
10. bekér két egész számot, majd kiírja az egészosztás maradékát (első per második)!
11. bekér két valós számot, majd kiírja az összegüket!
12. bekér két valós számot, majd kiírja a különbségüket (elsőből a második)!
13. bekér két valós számot, majd kiírja a szorzatukat!

C feladatok III

14. bekér két valós számot, majd kiírja a hányadosukat (első per második)!
15. bekér két egész számot, majd kiírja a valós hányadosukat (első per második)!
16. az oldalhosszból kiszámítja egy négyzet kerületét és területét!
17. a két oldalhosszból kiszámítja egy téglalap kerületét és területét!
18. a három oldalhosszból kiszámítja egy téglatest felszínét és térfogatát!
19. az átló hosszából kiszámítja egy négyzet kerületét és területét!
20. a sugárból kiszámítja egy kör kerületét és területét!
21. három oldalhosszból kiszámítja egy háromszög kerületét és területét!
22. a két adatból kiszámítja egy négyzet alapú "egyenes" gúla felszínét és térfogatát!
23. a két adatból kiszámítja egy "egyenes" kúp felszínét és térfogatát!

C feladatok IV

24. egy általad választott adatból kiszámítja egy tetraéder felszínét és térfogatát!
25. egy általad választott adatból kiszámítja egy hexaéder felszínét és térfogatát!
26. egy általad választott adatból kiszámítja egy oktaéder felszínét és térfogatát!
27. egy általad választott adatból kiszámítja egy ikozaéder felszínét és térfogatát!
28. egy általad választott adatból kiszámítja egy dodekaéder felszínét és térfogatát!
29. kiszámítja, hogy egy egyenletes sebességgel egyenes vonalban haladó test mennyi idő alatt tesz meg egy adott útszakaszt! Az input a sebesség és az úthossz.

C feladatok V

30. kiszámítja, hogy egy egyenletes sebességgel egyenes vonalban haladó test mekkora utat tesz meg adott idő alatt! Az input a sebesség és az eltelt idő.
31. kiszámítja, hogy egy adott utat adott idő alatt megtevő test mekkora átlagsebességgel halad! Az input a úthossz és az eltelt idő.
32. kiszámítja, hogy egy álló helyzetből egyenletesen gyorsuló, egyenes vonalban haladó test milyen távol lesz a kiindulási ponttól adott idő eltelte után! Az input a gyorsulás és az eltelt idő.
33. kiszámítja, hogy egy álló helyzetből egyenletesen gyorsuló, egyenes vonalban haladó test mennyi idő alatt tesz meg adott távolságot! Az input a gyorsulás és a megtett út.
34. kiszámítja egy álló helyzetből egyenletesen gyorsuló, egyenes vonalban haladó test gyorsulását, ha az adott idő alatt adott távolságot tesz meg! Az input a megtett út és az eltelt idő.

C feladatok VI

35. kiszámítja, hogy egy adott kezdősebességgel függőlegesen kilőtt test adott nehézségi gyorsulás ($g = 1,63 \frac{m}{s^2}$) mellett mennyi idő alatt esik vissza a Hold felszínére? Az input a kezdősebesség. Feltételezhető, hogy a kezdősebesség nem elég nagy ahhoz, hogy a testre ható tömegvonzás érezhetően megváltozzon.
36. adott nehézségi gyorsulás ($g = 9,81 \frac{m}{s^2}$) mellett a kilövési szög és a kezdősebesség alapján kiszámolja, hogy hol lesz a kilőtt test a felhasználó által megadott idő múlva. Számítsd ki azt is, hogy mikor és hol éri el a röppálya maximális magasságát. Nem kell számolnod a légellenállással és feltételezd, hogy a terep sík, és a megadott idő alatt a test még nem esik vissza a földre.

C függvények - az e konstans

Feladat: Írj egy függvényt, aminek nincs paramétere, és visszaadja e értékét 4 tizedesjegy pontossággal (2.7182). Írasd ki ezt az értéket!

Az e konstans értékének kiírása (e.c)

```
#include <stdio.h>

float e() {
  return 2.7182;
}

int main() {
  printf("e = %f\n", e());
  return 0;
}
```

C függvények - egy paraméteres függvény

Feladat: Írj egy függvényt, ami a paraméterül adott valós szám négyzetét kiírja!

Egy szám négyzetének kiírása (param.c)

```
#include <stdio.h>

float negyzet(float szam) {
  return szam * szam;
}
int main() {
  float valos_szam;
  printf("Kérek egy valós számot: ");
  scanf("%f", &valos_szam);
  printf("A szám négyzete: %f\n", negyzet(valos_szam));
  return 0;
}
```

C függvények - két paraméteres függvény

Feladat: Írj egy függvényt, ami összead két egész értéket, és visszatér az eredménnyel! Írj egy programot is, ami felhasználja ezt!

Két szám összegének kiírása (ossz.c)

```
#include <stdio.h>

int osszeg(int a, int b) {
  return a + b;
}

int main() {
  printf("3 + 8 = %d\n", osszeg(3, 8));
  return 0;
}
```

C függvények - függvény deklaráció I

Feladat: A programot rendezd át úgy, hogy előrébb legyen a main függvény definíciója mint az összeadó függvényé! (ossz2.c)

```
#include <stdio.h>

int osszeg(int, int);

int main() {
  printf("3 + 8 = %d\n", osszeg(3, 8));
  return 0;
}

int osszeg(int a, int b) {
  return a + b;
}
```

C függvények - függvény deklaráció II

Íly módon egy függvényt deklarálhatunk, és előbb csak később definiálni, hogy mit is csinál.

C globális és lokális változók I

Tekintsük az alábbi programot! (globals.c)

```
#include <stdio.h>

int globalis = 0;

int fuggveny(int parameter) {
  int lokalis = 0;
  lokalis += parameter;
  globalis += parameter;
  return lokalis;
}

int main() {
  int i;
  scanf("%d", &i);
```

C globális és lokális változók II

```
  printf("lokalis == %d\nglobalis == %d\n", fuggveny(i),
    globalis);
  scanf("%d", &i);
  printf("lokalis == %d\nglobalis == %d\n", fuggveny(i),
    globalis);
  scanf("%d", &i);
  printf("lokalis == %d\nglobalis == %d\n", fuggveny(i),
    globalis);
  return 0;
}
```

Feladat: Próbáljuk ki, fordul-e a program, ha a main függvényben megpróbáljuk felhasználni a lokalis változót!
Feladat: Mi történik, ha a globalis-t csak a fuggveny után deklaráljuk?

C függvény feladatok I

Adott C program

```
#include <stdio.h>

int muvelet(int, int);

int main() {
  int a, b;
  printf("Kérek két egész számot: ");
  scanf("%d %d", &a, &b);
  printf("Az eredmény: %d\n", muvelet(a, b));
  return 0;
}
```

C függvény feladatok II

Feladat: Egészítsd ki a muvelet függvény definíciójával úgy, hogy a program által kiírt eredmény

1. a két szám összege legyen!
2. a két szám különbsége legyen (elsőből a második)!
3. a két szám szorzata legyen!
4. a két szám egészosztás szerinti hányadosa legyen (első per második)!
5. a két szám egészosztásának maradéka legyen (első per második)!

C függvény feladatok III

Adott C program

```
#include <stdio.h>

float muvelet(float a, float b) {
  ...
}

int main() {
  float x, y;
  printf("Kérek két valós számot: ");
  scanf("%f %f", &y, &x);
  printf("Az eredmény: %f\n", muvelet(x, y));
  return 0;
}
```

C függvény feladatok IV

Írd meg a muvelet függvényben kipontozott részt úgy, hogy a program által kiírt eredmény Feladat: Egészítsd ki a muvelet függvény definíciójával úgy, hogy a program által kiírt eredmény

6. a két szám összege legyen!
7. a két szám különbsége legyen (elsőből a második)!
8. a két szám szorzata legyen!
9. a két szám hányadosa legyen (első per második)!

C függvény feladatok V

Feladat: Írd meg az alábbi programokat C nyelven úgy, hogy az adatok beolvasása és kiírása a main függvényben, de a számolások külön függvény(ek)ben történjenek.

1. Írj egy programot ami az oldalhosszból kiszámítja egy négyzet kerületét és területét!
2. Írj egy programot ami a két oldalhosszból kiszámítja egy téglalap kerületét és területét!
3. Írj egy programot ami a három oldalhosszból kiszámítja egy téglatest felszínét és térfogatát!
4. Írj egy programot ami az átló hosszából kiszámítja egy négyzet kerületét és területét!
5. Írj egy programot ami a sugárból kiszámítja egy kör kerületét és területét!
6. Írj egy programot ami három oldalhosszból kiszámítja egy háromszög kerületét és területét!

C függvény feladatok VI

7. Írj egy programot ami a két adatból kiszámítja egy négyzet alapú "egyenes" gúla felszínét és térfogatát!
8. Írj egy programot ami a két adatból kiszámítja egy "egyenes" kúp felszínét és térfogatát!
9. Írj egy programot ami egy általad választott adatból kiszámítja egy tetraéder felszínét és térfogatát!
10. Írj egy programot ami egy általad választott adatból kiszámítja egy hexaéder felszínét és térfogatát!
11. Írj egy programot ami egy általad választott adatból kiszámítja egy oktaéder felszínét és térfogatát!
12. Írj egy programot ami egy általad választott adatból kiszámítja egy ikozaéder felszínét és térfogatát!
13. Írj egy programot ami egy általad választott adatból kiszámítja egy dodekaéder felszínét és térfogatát!

C függvény feladatok VII

14. Írj egy programot ami kiszámítja, hogy egy egyenletes sebességgel egyenes vonalban haladó test mennyi idő alatt tesz meg egy adott útszakaszt! Az input a sebesség és az úthossz.
15. Írj egy programot ami kiszámítja, hogy egy egyenletes sebességgel egyenes vonalban haladó test mekkora utat tesz meg adott idő alatt! Az input a sebesség és az eltelt idő.
16. Írj egy programot ami kiszámítja, hogy egy adott utat adott idő alatt megtevő test mekkora átlagsebességgel halad! Az input a úthossz és az eltelt idő.
17. Írj egy programot ami kiszámítja, hogy egy álló helyzetből egyenletesen gyorsuló, egyenes vonalban haladó test milyen távol lesz a kiindulási ponttól adott idő eltelte után! Az input a gyorsulás és az eltelt idő.

C függvény feladatok VIII

18. Írj egy programot ami kiszámítja, hogy egy álló helyzetből egyenletesen gyorsuló, egyenes vonalban haladó test mennyi idő alatt tesz meg adott távolságot! Az input a gyorsulás és a megtett út.
19. Írj egy programot ami kiszámítja egy álló helyzetből egyenletesen gyorsuló, egyenes vonalban haladó test gyorsulását, ha az adott idő alatt adott távolságot tesz meg! Az input a megtett út és az eltelt idő.
20. Írj egy programot ami kiszámítja, hogy egy adott kezdősebességgel függőlegesen kilőtt test adott nehézségi gyorsulás ($g = 1,63 \frac{m}{s^2}$) mellett mennyi idő alatt esik vissza a Hold felszínére? Az input a kezdősebesség. Feltételezhető, hogy a kezdősebesség nem elég nagy ahhoz, hogy a testre ható tömegvonzás érezhetően megváltozzon.

C függvény feladatok IX

21. Írj egy programot ami adott nehézségi gyorsulás ($g = 9,81 \frac{m}{s^2}$) mellett a kilövési szög és a kezdősebesség alapján kiszámolja, hogy hol lesz a kilőtt test a felhasználó által megadott idő múlva. Számítsd ki azt is, hogy mikor és hol éri el a röppálya maximális magasságát. Nem kell számolnod a légellenállással és feltételezd, hogy a terep sík, és a megadott idő alatt a test még nem esik vissza a földre.

Feladatok

További feladatok a `/pub/progalap/Gyakorlat/gyak04/` alatt.

C operátorok (nem teljes lista)

==	Egyenlő
!=	Nem egyenlő
<	Kisebb
>	Nagyobb
<=	Kisebb, vagy egyenlő
>=	Nagyobb, vagy egyenlő
!	Tagadás
||	Logikai vagy
&&	Logikai és

A 0 számot hamisnak, a nem nulla számokat igaznak tekintjük.

Egy szám paritásának eldöntése

Feladat: Készíts egy programot, ami bekér egy egész számot és kiírja, hogy az adott szám páros vagy páratlan-e.

Egy szám paritásának eldöntése (paros.c)

```
#include <stdio.h>
int main() {
  int x;
  printf("Kérek egy egész számot:");
  scanf("%d", &x);

  if (x % 2 == 0)
    printf("A megadott szám páros.\n");
  else
    printf("A megadott szám páratlan.\n");
  return 0;
}
```

Egy szám oszthatóságának eldöntése I

Feladat: Módosítsuk most a programot úgy, hogy két egész számot kérjen be a program majd írja ki, hogy az első szám osztható-e a másodikkal (osztoja.c)!

```
#include <stdio.h>
int main() {
  int x, y;

  printf("Kérek egy egész számot:");
  scanf("%d", &x);
  printf("Kérek egy másik egész számot:");
  scanf("%d", &y);

  if (x % y != 0) {
    printf("%d nem osztója %d-nek.", y, x);
  } else {
```

Egy szám oszthatóságának eldöntése II

```
    printf("%d osztója %d-nek.", y, x);
  }

  return 0;
}
```

Próbáljuk ki, mi történik, ha a második szám 0!

Egy szám oszthatóságának eldöntése, javítva I

Feladat: Javítsuk ki az előző programot (osztoja.c)!

```
#include <stdio.h>
int main() {
  int x, y;

  printf("Kérek egy egész számot:");
  scanf("%d", &x);
  printf("Kérek egy másik egész számot:");
  scanf("%d", &y);

  if (y == 0) {
    printf("Nullával nem osztunk!\n");
  } else {
    if (x % y != 0) {
      printf("%d nem osztója %d-nek.", y, x);
```

Egy szám oszthatóságának eldöntése, javítva II

```
    } else {
      printf("%d osztója %d-nek.", y, x);
    }
  }

  return 0;
}
```

Házi Feladat: Módosítsuk úgy az előző programot, hogy ez az oszthatósági vizsgálat egy függvényen belül legyen. Ha a két szám osztója egymásnak az fgv visszatérési értéke legyen 1, különben pedig 0. A 0-val osztást kezeljük egyszerűen nem osztóként.

Egy szám oszthatóságának eldöntése, if nélkül I

Feladat: Írjuk meg if nélkül a fenti programot! (esvagy.c)

```
#include <stdio.h>
int main() {
  int x, y;

  printf("Kérek egy egész számot:");
  scanf("%d", &x);
  printf("Kérek egy másik egész számot:");
  scanf("%d", &y);

  (y != 0) || printf("Nullaval nem osztunk!\n");
  (y != 0) && (x % y == 0) &&
              printf("%d osztója %d-nek.", y, x);
  (y != 0) && (x % y != 0) &&
              printf("%d nem osztója %d-nek.", y, x);
```

Egy szám oszthatóságának eldöntése, if nélkül II

```
  return 0;
}
```

Egy szám paritásának eldöntése, feltételes kifejezéssel I

Feladat: Írjuk ki egyetlen printf segítségével, hogy egy szám páros vagy páratlan-e! (paros_cond.c)

```
#include <stdio.h>
int main() {
  int x;

  printf("Kérek egy egész számot: ");
  scanf("%d", &x);

  printf("A szám %s.\n", (x % 2 == 0) ? "páros" :
                                        "páratlan");

  return 0;
}
```

Egy szám paritásának eldöntése, feltételes kifejezéssel II

Házi feladat: Írjuk ki egyetlen printf használatával, hogy egy szám osztója-e egy másiknak!

Egy szám paritásának eldöntése, egymásba ágyazott feltételes kifejezéssel

A feltételes kifejezéseket egymásba is ágyazhatjuk.

```
#include <stdio.h>
int main () {
  int x;

  printf("Kérek egy egész számot: ");
  scanf("%d", &x);
  printf("Kérek egy másik számot: ");
  scanf("%d", &y);

  printf("Osztója-e %d-nek %d?. %s\n", x, y,.
    (y == 0) ? "A kerdes ertelmetlen!" :
    ((x % y == 0) ? "Igen, osztója." :
                    "Nem, nem osztója.")
  );
```

C feltételes elágazás(switch)

Feladat: írjunk egy függvényt, ami egy `x` egész számot kap paraméterként és kiírja, hogy a hét `x`. napja milyen nap.

```
void hetnapja_if (short int x) {
  if (x==1) {
    printf("Hétfő\n");
  } else if (x==2) {
    printf("Kedd\n");
  } else if (x==3) {
    printf("Szerda\n");
  } else if (x==4) {
    printf("Csütörtök\n");
  } else if (x==5) {
    printf("Péntek\n");
  } else if (x==6) {
    printf("Szombat\n");
  } else if (x==7) {
    printf("Vasárnap\n");
```

C feltételes elágazás(switch)

Feladat: írjuk meg ugyanezt a függvényt switch használatával!

```
void hetnapja_switch (short int x) {
  switch (x) {
    case 1:.
      printf("Hétfő\n");
      break;
    case 2:
      printf("Kedd\n");
      break;
    case 3:
      printf("Szerda\n");
      break;
    case 4:
      printf("Csütörtök\n");
      break;
    case 5:
      printf("Péntek\n");
```

C feltételes elágazás(switch)

Feladat: írjunk egy függvényt, ami egy `x` egész számot kap paraméterként és kiírja, hogy a hét `x`. napja milyen nap!

```
void hetnapja_if (short int x) {
  if (x == 1) {
    printf("Hétfő\n");
  } else if (x == 2) {
    printf("Kedd\n");
  } else if (x == 3) {
    printf("Szerda\n");
  } else if (x == 4) {
    printf("Csütörtök\n");
  } else if (x == 5) {
    printf("Péntek\n");
  } else if (x == 6) {
    printf("Szombat\n");
  } else if (x == 7) {
    printf("Vasárnap\n");
```

C feltételes elágazás(switch)

- Kérdés: Hogyan kellene módosítani a függvényt akkor, ha számok helyett a napok kezdőbetűit szeretnénk használni?
- Kérdés: Működne-e ugyanez akkor, ha egész v. karakter típusú változó helyett pl. `float` vagy `double` típusú változó lenne a switch feltételében?
- Feladat: Módosítsd a paraméter típusát `unsigned int`-ről `float`-ra, készíts egy main fgvt, ami meghívja a `hetnapja_switch(4.0)`-t!
- Feladat: Nézzük meg mi történik, akkor ha elhagyjuk a csütörtök és a péntek napok után a `break` utasítást. a korábban elkészített main függvényünkben hívjuk most meg a `hetnapja_switch` függvényt a `4,5` és 6 paraméterekkel!

C ciklus(while)

Feladat: írjunk egy programot, ami kiírja 1-től 10-ig számokat!

Számok kiíratása 1-től 10-ig (while1.c)

```
#include <stdio.h>
int main() {
  int i = 1;
  while (i <= 10) {
    printf("%d\n", i++);
  }
  return 0;
}
```

C ciklus(while)

Feladat: Írjunk olyan prgramot, ami addig kér be számokat a billentyűzetről, amíg a beírt szám nem 0! (0 az adott végjel)

Számok kiíratása 1-től 10-ig, végjellel (while2.c)

```
#include <stdio.h>
int main() {
  int x;

  printf("Kérek egy számot (kilépéshez: 0):");.
  scanf("%d", &x);
  while (x != 0) {
    printf("Kérek egy számot (kilépéshez: 0):");.
    scanf("%d", &x);
  }
  return 0;
}
```

C ciklus(while)

Feladat: Módosítsuk a programot úgy, hogy végeredményként írja ki a beírt számok összegét! (while3.c)

```
#include <stdio.h>
int main() {
  int x;
  int osszeg = 0;

  printf("Kérek egy számot (kilépéshez: 0):");
  scanf("%d", &x);
  while (x != 0) {
    osszeg += x;
    printf("Kérek egy számot (kilépéshez: 0):");
    scanf("%d", &x);
  }
  printf("A számok összege: %d\n", osszeg);
  return 0;
}
```

C ciklus(while)

Feladat: Módosítsuk a programot úgy, hogy végeredményként írja ki a beírt számok összegét! (while4.c)

```
#include <stdio.h>
int main() {
  int x;
  int osszeg = 0;

  while (1) {
    printf("Kérek egy számot (kilépéshez: 0):");
    scanf("%d", &x);

    if (x == 0) {
      break;
    } else {
      osszeg += x;
    }
}
```

C ciklus(do-while)

Feladat: Írjunk egy olyan programot `do-while` ciklus segítségével, ami 0 végjelig kér be számokat, majd kírja azok összegét. A ciklusban ne szerepeljen a `break` utasítás! (dowhile.c)

```
#include <stdio.h>
int main() {
  int x;
  int osszeg = 0;

  do {
    printf("Kérek egy számot (kilépéshez: 0):");
    scanf("%d", &x);
    osszeg += x;
  } while (x != 0);
  printf("A számok összege: %d\n", osszeg);
  return 0;
}
```

C ciklus(for)

Feladat: Írjunk egy programot, ami összeszorozza 1-10-ig a számokat! (for1.c)

```
#include <stdio.h>
int main() {
  int i;
  int szorzat;

  for (i=1, szorzat=1; i <= 10; ++i) {
    szorzat *= i;
  }

  printf("A számok szorzata: %d\n", szorzat);
  return 0;
}
```

C ciklus(for)

Feladat: Hogyan nézne ki ugyanez a program while ciklussal? (for2.c)

```
#include <stdio.h>
int main() {
  int i;
  int szorzat;

  i = 1;
  szorzat = 1;
  while (i <= 10) {
    szorzat *= i;
    ++i;
  }

  printf("A számok szorzata: %d\n", szorzat);
  return 0;
}
```

C ciklus(for)

Feladat: Módosítsuk a for ciklust úgy, hogy csak minden 3-mal osztható számot szorozzon össze! (for3.c)

```
#include <stdio.h>
int main() {
  int i;
  int szorzat;

  for (i = 3, szorzat = 1; i <= 10; i += 3)
    szorzat *= i;

  printf("A számok szorzata: %d\n", szorzat);
  return 0;
}
```

C ciklus(for)

- Feladat: Próbáljuk ki mit csinál az alábbi for ciklus:

```
int i, j, out;
for (i = 1, j = 100, out = 0; i <= 10; i++, j--)
    out += i * j;
```

- Feladat: Módosítsuk a ciklusmagot úgy, hogy egy printf segítségével kiírjuk az i,j és out aktuális értékét!
- Kérdés: Mi a , művelet eredménye? a=(1,2,3,4) ?
- Kérdés: Mit csinál az a = 4, b = a + 3, c = b * 2 + 1; utasítás? Melyik kifejezésket értékeli ki, milyen sorrendben és mi lesz a kifejezés értéke?

Feladatok

További feladatok a `/pub/progalap/Gyakorlat/gyak05/` alatt.

C preprocesszor I

A C preprocesszor behelyettesíti a makróinkat, betölti a fejléc fileokat. A `gcc -E program.c` parancs kimenete a `program.c` C forrásfájl preprocesszált változatát adja.
Feladat: Írj egy programot, ami 1-től 10-ig kiírja a számokat, majd minden második, majd minden negyedik számot! (konstans.c)

```
#include <stdio.h>
int main() {
  int i;
  for(i=1; i<=10; i++) {
    printf(" %d", i);
  }
  putchar('\n');
  for(i=1; i<=10; i+=2) {
    printf(" %d", i);
  }
```

C preprocesszor II

```
  putchar('\n');
  for(i=1; i<=10; i+=4) {
    printf(" %d", i);
  }
  putchar('\n');
  return 0;
}
```

Feladat: Módosítsuk úgy a programot, hogy 21-től 144-ig írjon ki! Hány helyen kellett átírnunk számokat?
Feladat: Csináljuk meg ugyanezt konstansokkal! Így hány helyen kellene módosítani? (konstans2.c)

C preprocesszor III

```
#include <stdio.h>

#define A 1
#define B 10

int main() {
  int i;
  for(i=A; i<=B; i++) {
    printf(" %d", i);
  }
  putchar('\n');
  for(i=A; i<=B; i+=2) {
    printf(" %d", i);
  }
  putchar('\n');
  for(i=A; i<=B; i+=4) {
```

C preprocesszor IV

```
    printf(" %d", i);
  }
  putchar('\n');
  return 0;
}
```

Nézzük meg, mi lesz a preprocesszálás eredménye!

```
gcc -E konstans2.c > konstans2.i
gcc konstan2s.i -o konstans2
```

A `konstans2.i` gyakorlatilag az `stdio.h` fájllal(és az onnan beinclude-olt fájlokkal) kezdődik, a saját kódunk a fájl végén van. Látható, hogy az összes `A` helyére 1 került, az összes `B` helyére pedig 2. A preprocesszor nem végez szintaktikai ellenőrzést, csupán behelyettesít.

C preprocesszor, hiba I

Feladat: Hol jelez hibát a fordító a preproc.c -ben? Miért ott?

Fordítsuk és futtassuk a fájlt! (preproc.c)

```
#include <stdio.h>
#define int 100.0
int main() {
  float f = int;
  printf("%f\n", f);
  return 0;
}
```

Magyarázat: A `main()` előtt levő `int` lecserélődik `100.0`-ra, ez okozza a szintaktikai hibát.

C enum, szintaktika I

Az `enum` (felsorolás) szerkezet arra jó, hogy bizonyos szöveges nevekhez konstans egész számokat rendeljünk. Ezzel egyfajta típust hozunk létre. Ha nem adunk meg mást, akkor az első név a 0 (int) értéket kapja, a következők mindig eggyel többet. Konkrét értéket az egyenlőséggel adhatunk.
Pl.

```
enum Szin { Kor, Pikk, Treff, Karo};
```

Ekkor Kor=0, Pikk=1, Treff=2, Karo=3, azonban

```
enum Szin { Kor, Pikk, Treff=5, Karo};
```

esetén Kor=0, Pikk=1, Treff=5, Karo=6.

C enum, hét napjai I

Feladat: Definiálj egy felsorolástípust a hét napjainak tárolására, majd írasd ki a napok értékeit! (enum.c)

```
#include <stdio.h>
int main(){
        enum het { Hetfo,
                   Kedd,
                   Szerda,
                   Csutortok,
                   Pentek,
                   Szombat,
                   Vasarnap
                 } nap;
        for(nap = Hetfo; nap <= Vasarnap; nap++) {
                printf("%d\n", nap);
        }
```

C enum, hét napjai II

```
        return 0;
}
```

Feladat: Mi történik, ha `Hetfo=1` -ként adod meg az első elemet?
Feladat: Mi történik, ha `Szombat=10` -ként adod meg a hatodik elemet?
Feladat: Adhatod-e az enum mindegyik elemének ugyanazt az int értéket?
Feladat: Készíts egy függvényt, ami megadja a hét következő napját!

```
enum het kovetkezo(enum het n) {
        if(n==Vasarnap) {
                return Hetfo;
        }
        return n+1;
}
```

C enum, hét napjai III

Tekintsük a következő kódrészletet:

```
enum het nap;
for(nap=Hetfo; nap <= Vasarnap; nap=kovetkezo(nap)) {
        printf("%d\n", nap);
}
```

Mi lesz az eredménye, és miért?

C enum, alma.c I

Feladat: Tekintsük a következő példa programkódot: (alma.c)

```
#include <stdio.h>
int main() {
  enum het { Hetfo, Kedd, Szerda, Csutortok, Pentek, Szombat,
  typedef enum { piros, zold, sarga } colors;
  colors col;

  printf("Milyen napon szeretnél almát enni? "); scanf("%d",&n
  printf("Milyen színű almát szeretnél enni? "); scanf("%d",&c
  switch(nap) {
    case Hetfo :
    case Kedd :
    case Szerda :
    case Csutortok :
    case Pentek :
```

C enum, alma.c II

```
    printf("Csak hétvégén tudok almát felszolgálni!\n");
    break;
  case Szombat :
  case Vasarnap :
    printf("Mivel hétvége van, alma is van!\n");
    switch(col){
      case piros:
        printf("A piros alma egészséges, jó választás!\n");
        break;
      case zold:
        printf("Vigyázz, a zöldalma savanyú!\n");
        break;
      case sarga:
        printf("A sárga alma is nagyon finom!\n");
        break;
      default :
```

C enum, alma.c III

```
            printf("Nem ismerek ilyen színű almát!\n");
        }
        break;
      default:
        printf("A hét csak 7 napból áll!\n");
        break;
    }

    return 0;
}
```

C tömbök I

- A C programozási nyelv lehetőséget nyújt tömbök használatára.
- Tömbnek tekintjük azt az adatszerkezetet, amely *egyforma típusú* értékek előre meghatározott sorozatát jelöli.
- Gyakorlatilag a memóriában egymás után helyezkednek el a tömb elemei, és a tömb méretét nem tároljuk a memóriában (kivéve, ha dinamikusan allokáltuk).
- Ha deklarálunk egy tömböt, meg kell adnunk egy maximális elemszámot, ennél több elemet nem fog tudni tárolni a tömb.
- A tömb értékei is ugyan úgy inicializálatlanok, mint a változók értéke kezdetben.
- Egy n-elemű tömböt 0-tól $n-1$-ig indexelünk. Fontos, hogy ezeket a határokat szigorúan tartsuk be, mert a határokon túl található adatok módosítása érzékenyen érintheti a program futását.

C tömbök II

- A tömb deklarálásakor tudnunk kell, hogy milyen típusú adatot tárol, és hány darabot. Egy 10 elemű egészeket tároló tömböt tehát így deklarálunk:

```
int egesz_tomb[10];
```

- Ha kezdőértékeket szeretnénk megadni, akkor azokat fel kell sorolni kapcsos zárójelek között, vesszővel elválasztva:

```
int egesz_tomb[10] = {1, 2, 3, 4, 9, 8, 7, 6, 5, 10};
```

  Ebben az esetben megadtuk az elemszámot, amelynek egyeznie kell a kapcsos zárójelek közt felsorolt elemek számával.

- Amennyiben kezdőértékekkel adunk meg egy tömböt, az elemszámot elhagyhatjuk, ekkor a fordító az általunk megadott értékek számát tekinti elemszámank:

```
int egesz_tomb[] = {1, 2, 3, 4, 9, 8, 7, 6, 5, 10};
```

C tömbök III

- A tömb egy elemére úgy hivatkozunk, hogy leírjuk a tömb azonosítóját (változónév), majd szögletes zárójelek között megadjuk az indexet, pl. a

  `het_napjai[2]`

  kifejezés a `het_napjai` nevű tömb `2.` (`3.`) elemére hivatkozik. Az index lehet egy összetettebb kifejezés is, pl.

  `het_napjai[keres_nap(i * 2) % 7]`

C tömbök, feltöltés számokkal I

Feladat: Készíts egy 10 egész szám tárolására alkalmas tömböt! Töltsd fel az 1..10 értékekkel, majd írasd ki az elemeit!

```
#include <stdio.h>
#define N 10
#define M 10
int main(){
  int tomb[N];
  int i;
  for(i = 0; i < M; i++) {
    tomb[i] = i + 1;
  }
  for(i = 0; i < M; i++) {
    printf(" %d", tomb[i]);
  }
  printf("\n");
```

C tömbök, feltöltés számokkal II

```
  return 0;
}
```

Magyarázat:

- Lesz egy konstansunk, `N=10` és `M=10`.
- `N` adja meg, hogy mekkora lesz a tömbünk (a legnagyobb eleme $N-1$ indexű).
- `M` pedig megadja, hogy milyen elemnél kisebb elemeket szúrjunk be a tömbbe (a legnagyobb indexnél eggyel nagyobb szám).

Feladat: Próbáljuk ki, hogy mi történik, ha túlindexeljük a tömböt!

C tömbök, kétdimenziós tömb I

Feladat: Készíts egy 3x3-as mátrixot, töltsd fel elemekkel, majd írasd ki az elemeit sor illetve oszlopfolytonosan is! (tomb2d.c)

```
#include <stdio.h>
#define N 3

int main(){
  int tomb[N][N];
  int i, j;
  for(i = 0; i < N; i++) {
    for(j = 0; j < N; j++) {
      scanf("%d", &(tomb[i][j]));
    }
  }
  for(i = 0; i < N; i++) {
    for(j = 0; j < N; j++) {
```

C tömbök, kétdimenziós tömb II

```
      printf("%d", tomb[i][j]);
    }
  }
  for(i = 0; i < N; i++) {
    for(j = 0; j < N; j++) {
      printf("%d", tomb[j][i]);
    }
  }
  return 0;
}
```

C tömbök, kétdimenziós tömb III

Magyarázat:

- Deklarálunk egy NxN-es egész típusú tömböt(N=3):

  ```
  int tomb[N][N];
  ```

- Ez azt jelenti, hogy van egy egésztömb típusú tömbünk. A tömbünknek több dimenziója is lehetne, pl. 3.
- Először feltöltjük a tömböt egész számokkal. Ezt két egymásba ágyazott *for ciklus*sal érjük el. Külön számlálót hsználunk az oszlophoz és a sorhoz: `i` és `j`.
- Figyeljük meg a beolvasás sorát:

  ```
  scanf("%d", &(tomb[i][j]));
  ```

C tömbök, kétdimenziós tömb IV

- Ezután kiíratjuk a tömb elemeit sorfolytonosan, majd oszlopfolytonosan. Figyeljük meg, hogy a kiíratást végző két ciklus csak a tömbre hivatkozásban különbözik:

  `tomb[i][j]`

  és

  `tomb[j][i]`

C karaktertömbök (sztringek) I

- A C nyelv nem rendelkezik `sztring` típússal, azonban egy szöveges adat ábrázolására karaktertömböket használunk.
- Egy sztring tehát valójában `char[]` típusú.
- A sztringeknek speciális lezáró karaktere van, C-ben `'\0'` (char), `0` (int) jelölést használhatjuk ezen nem nyomtatható karakter szimbolizálásához.
- Ez azt jelenti, hogy (majdnem) minden sztring egy ilyen NUL karakterre végződik, amely nem része a sztringnek, de a tömbnek igen. Ezért ilyenkor a tömb méretét a szöveg hosszánál eggyel nagyobb méretűre kell deklarálni.
- Amennyiben fix méretű szövegekkel dolgozunk, megjegyezhetjük a méretet, és nem kell lezáró NUL karaktert használni.
- A sztringeket megadhatjuk idézőjelek közt is,

```
char szoveg[] = "Hello";
```

C karaktertömbök (sztringek) II

illetve tömbös módon is,

```
char szoveg2[] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

C sztring függvények I

```
#include <string.h>
```

`size_t strlen(const char *s);` Megadja a szöveg hosszát (lezáró 0 nélkül).

`char *strstr(const char *haystack, const char *needle);` Megadja egy részsztring pozícióját.

`char *strcpy(char *dest, const char *src);` `src` sztringet átmásolja dest-be, lezáró nullával. Fontos, hogy legyen elég hely a `dest` sztringben.

`char *strncpy(char *dest, const char *src, size_t n);` Hasonlóan, de csak az első *n* karaktert (ha nincs ezek között lezáró NUL, akkor a `dest` sztringbe sem íródik).

`char *strcat(char *dest, const char *src);` Az `src` sztringet átmásolja a `dest` sztringbe (lezáró nullával).

`char *strncat(char *dest, const char *src, size_t n);` Hasonlóan, de csak az első *n* karaktert (plusz a lezáró NUL-t).

C sztring függvények II

`char *strchr(const char *s, int c);` Az `s` sztring első olyan pozícióra ad mutató, ahol a `c` karakter megtalálható.

`char *strrchr(const char *s, int c);` Az `s` sztring első olyan pozícióra ad mutató, ahol a `c` karakter megtalálható.

Házi feladat: valósítsuk meg ezeket a függvényeket a `string.h` használata nélkül! Érdemes először az strlen()-t megcsinálni. `char*` helyett használható `char[]`.

C sztring feladatok I

Feladat: Deklarálj egy megfelelő hosszúságú karaktertömböt (`str`), majd írd bele a `Hello Vilag!` szöveget! (sztring.c)

```
#include <stdio.h>
int main(){
  char str[13] = "Hello Vilag!";
  printf("%s\n", str);
  return 0;
}
```

C sztring feladatok I

Feladat: Módosítsd a programot úgy, hogy a következő sorba csak a `Hello` szöveget írja ki! (sztring2.c)

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

int main(){
  char str[13];
  strcpy(str, "Hello Vilag!");
  printf("%s\n", str);
  str[5] = '\0';
  printf("%s\n", str);
  return 0;
}
```

C sztring feladatok II

Házi Feladat: Mi történik, ha az előző végjelet('\0') visszacseréled ' ' karakterre?
Házi Feladat: Mi történik, ha az `str` méretét leveszed mondjuk 4-re?

C sztring feladatok I

Feladat: Írasd ki a teljes `str` tömb karaktereit! (sztring3.c)

```
#include <stdio.h>
int main(){
  char str[13]="Hello Vilag!";
  int i;
  for(i = 0; i < 13; i++) {
    putchar(str[i]);
  }
  putchar('\n');
  return 0;
}
```

C sztring feladatok II

Feladat: Írasd ki a teljes str tömb karaktereit úgy, hogy a méretet nem határozod meg explicit módon! (sztring4.c)

```
#include <stdio.h>
#include <string.h>
int main(){
  char str[]="Hello Vilag!";
  int i;
  for(i = 0; i < strlen(str); i++) {
    putchar(str[i]);
  }
  putchar('\n');
  return 0;
}
```

C tömb feladatok I

Feladat: Írj egy függvényt, ami egy egész tömböt kap paraméterül és lecseréli benne az elemeket az abszolútértékükre. A tömb kiírását szintén függvény végezze! (tombfgv.c)

```
#include <stdio.h>
#define N 10
void tombabs(int tomb[], int meret) {
        int i;
        for(i = 0; i < meret; i++) {
                if(tomb[i] < 0) {
                        tomb[i] = -tomb[i];
                }
        }
}
void kiir(int tomb[], int meret) {
        int i;
```

C tömb feladatok II

```
        for(i = 0; i < meret; i++) {
                printf(" %d", tomb[i]);
        }
        putchar('\n');
}
int main(){
        int i, T[N], e = 1;
        for(i = 0; i < N; i++) {
                T[i] = e;
                e *= -2;
        }
        kiir(T, N);
        tombabs(T, N);
        kiir(T, N);
        return 0;
}
```

Feladatok

További feladatok a `/pub/progalap/Gyakorlat/gyak06/` alatt.

Az egész típus

- Az egész számokat szokás binárisan tárolni. Ez azt jelenti, hogy a számunkat n biten tároljuk, így egy ekkora tárhelyen 2^n különböző értéket tárolhatunk.
- Ekkor az egyes bitek rendre $2^0 = 1$, $2^1 = 2$, $2^2 = 4$, ..., 2^n decimális értékeket reprezentálnak.
- Megegyezés szerint az ábrázolásunk lehet

  kis endián (little endian) a legkisebb helyiértékű bit az első bit

  nagy endián (azaz big endian) a legnagyobb helyiértékű bit az első bit
- Amennyiben csak nemnegatív pozitív értékeket szeretnénk tárolni, szokás egy n bites számot a $[0; 2^n - 1]$ intervallumra képezni.
- Amennyiben negatív számokat is szeretnénk tárolni, elhatározzuk, hogy a legnagyobb helyiértékű bitünk egy előjelbit lesz. Ha az előjelbit 1, akkor negatív számról beszélünk, különben pedig nemnegatív számról. A $[-\frac{2^n}{2}; -\frac{2^n}{2} - 1]$ intervallumot szokás használni.

Az egész típus a C nyelvben I

- Az egész típus neve a C nyelvben az `int`.
- Az `int` típus hagyományosan 4 byte-on (32 biten) tárolódik, de ez architektúra-függő.
- A `limits.h` fejléc állomány tartalmaz a cél architektúrára vonatkozó korlátokat, `int`-ekre specifikusan:

  INT_MAX A legnagyobb érték, amit egy `int` képes tárolni.

  INT_MIN A legkisebb érték, amit egy `int` képes tárolni.

- Vigyáznunk kell a korlátoknál:

  túlcsordulás Akkor fordul elő, ha túlhaladjuk `INT_MAX`-ot, és így újból `INT_MIN`-ről indulunk. pl. `INT_MAX+1`

  alulcsordulás Akkor fordul elő, ha alulhaladjuk `INT_MIN`-t, és így újból `INT_MAX`-ot kapjuk, pl. `INT_MIN-1`

- Ha egészet osztunk egésszel (a `/` jellel), akkor az mindig egészosztást jelent, és az eredmény is egy egész lesz, azaz pl. `6 / 2 == 3`, míg `5 / 2 == 2`

Az egész típus a C nyelvben II

Modifier-ek:

short	feleakkora méret
long	kétszer akkora méret
long long	négyszer akkora méret
unsigned	előjeltelen
signed	előjeles (alapértelmezett)

- A fentieket értelmesen kombinálhatjuk is, pl. `unsigned long long int` egy négyszeres méretű előjeltelen egészt jelent, vagy a `signed short int` kis méretű előjeles egészek tárolására alkalmas.
- A `long int` helyett írhatunk csak `long`-ot, hasonlóan `long long`-ot és `short`-ot. A modifierek sorrendje azonban kötött, így pl. `long unsigned int` nem megengedett.

Számrendszerek a C nyelvben

- A C nyelv alapvetően decimális számokkal dolgozik.
- Megadhatunk egy számot hexadecimális alakban is, pl. `0xA9` (=154).
- Megadhatunk egy számot oktális alakban is, ha 0-val kezdjük a számot, pl. `01237` (= `671`).
- A hexadecimális és oktális számok kapcsán elsősorban nemnegatív számokra gondolunk (azonban lebegőpontos is lehetséges).
- Nagy számokat érdemes minél nagyobb számrendszerben leírni, hiszen akkor kevesebb számjegyre van szükség, pl.
  `unsigned long long int x = 0xDEADBEEFCAFEFACE;`
- Házi feladat: hogyan tudunk *printf*-el egy számot hexadecimális alakban kiírni? (Tipp: `man 3 printf`)

Az egész típusok korlátai a C nyelvben I

Méret	Bitek száma	Tárolható értékek száma	Előjeles-e?	Intervallum	Max. dec. számjegyek száma
byte/char	8	256	unsigned	[0; 255]	3
			signed	[-128; 127]	3
short	16	65536	unsigned	[0; 65535]	5
			signed	[-32768; 32767]	5
int	32	4294967296	unsigned	[0; 4294967295]	10
			signed	[-2147483648; 2147483647]	10
long	32	4294967296	unsigned	[0; 4294967295]	10
			signed	[-2147483648; 2147483647]	10
long long	64	18446744073709551616	unsigned	[0; 18446744073709551615]	20
			signed	[-9223372036854775808; 9223372036854775807]	19
n-bites egész	n	2^n	előjeltelen	$[0; 2^n - 1]$	$\log_{10} 2^n$
			előjeles	$[-\frac{2^n}{2}; \frac{2^n}{2} - 1]$	$\log_{10} 2^{n-1}$

Az értékek egy átlagos *x86* architektúrájú számítógépre vonatkoznak.

Az egész típusok korlátai a C nyelvben II

- Ha egy nagy konstans számot akarunk a C programunkba írni, akkor explicit módon meg kell adnunk a típusát:

U	unsigned
L	long
LL	long long
UL	unsigned long
ULL	unsigned long long

- Például `unsigned long long int x = 429496729600ULL;`

A valós (lebegőpontos) típus

- A feladat az, hogy szeretnénk a számítógéppel valós számokkal számolni, ehhez viszont tudnunk kell azokat ábrázolni. Mivel véges tár áll rendelkezésre, és két különböző valós szám között végtelen sok másik valós szám van, ezért a valós számokat közelíteni fogjuk egy törtszámmal.
- A lebegőpontos számok ábrázolása során az IEEE 754 szabványnak megfelelően szokás eljárni. (ld. a szabvány weboldalát)
- A közelítés mértéke változó lehet

egyszeres	32 bit
kétszeres	64 bit
négyszeres	128 bit

A valós (lebegőpontos) típus a C nyelvben

- A C nyelv kétféle elemi valós típust használ (az *IEEE-754* szerint):
  - float (leginkább 32 bit)
  - double (leginkább 64 bit)
- A lebegőpontos számok használata során figyelembe kell vennünk, hogy veszíthetünk a számok pontosságából (ld. későbbi példa).
- Tegyük fel, hogy van két lebegőpontos, inicializált változónk (x és y, `floatok`). Ekkor nem lehetünk abban biztosak, hogy $x < y$, $x == y$, $x > y$ közül legalább egy teljesül.
- Egy konstanst számot lebegőpontos számnak értelmez a fordító, ha szerepel benne a `.` szimbólum (pl. `5.0`).
- Egy konstans számot explicit módon lebegőpontosként megadhatunk, ha a szám végére a `f` karaktert írjuk (pl. `5.0f`). Ez elméletileg gyorsíthatja a fordítást/futást.
- Használhatjuk az `1e10` alakot is, ami 10^{10}-t jelent.

Értékek konvertálása (castolás)

- A *cast*-olás során egy típusból megpróbálunk egy másik típust létrehozni.
- A kívánt cél típust zárójelek közé rakjuk, és a konvertálandó érték elé írjuk.
- Pl. `int x = (unsigned char) -1;` után az `x` értéke 255 lesz.

Saját típusok definiálása C-ben I

- C-ben az elemi típusok mellett definiálhatunk saját típusokat is, ezt a `typedef` kulcsszóval tesszük.
- A `typedef`-nek először meg kell mondani, hogy milyen típust szeretnénk használni, majd meg kell adni, hogy mi legyen az új típus neve:
  `typedef RÉGITIPUS ÚJTIPUS;`

Saját típusok definiálása C-ben II

Feladat: Hogyan tudunk létrehozni egy olyan vector nevű tömb típust, amely egy háromdimenziós térbeli vektort reprezentál?

```
typedef double vector[3];
```

Feladat: Hogyan lehet létrehozni egy N hosszúságú sztringek tárolására szolgáló karaktertömb típust?

```
#define N 100
typedef char string[N+1];
```

Feladat: Hozz létre külön típust 16 bites nemnegatív értékek tárolására!

```
typedef unsigned short int uint_16;
```

A sizeof() operátor I

- A C nyelv tartalmaz egy `sizeof` operátort, amellyel lekérdezhetjük fordítási időben egy tetszőleges adatszerkezet méretét.
- Ha tömb méretét kérdezzük le, akkor a tömb teljes memóriterületének méretét megkapjuk, ezt leosztva pl. a 0. elem méretével, megkapjuk a tömb elemszámát.
- A `sizeof()` egy előjeltelen (`size_t` típusú) értéket ad, értelemszerűen.
- A `sizeof()` a dinamikus memória használatnál jól fog jönni.

A sizeof() operátor II

Feladat: Hány bájton tárolódik a char típus?

char méretének megállapítása (meret-char.c)

```
#include <stdio.h>
int main(){
  printf("char: %zu\n", sizeof(char));
  return 0;
}
```

Megjegyzés: A `z` formátumkarakter a `size_t` típusra utal, az `u` pedig előjeltelen számra, ezeket illik használni `sizeof()`-al.

Típussal kapcsolatos feladatok (char) I

Feladat: Írasd ki a 64 és 95 közé eső kódú karaktereket!

Karakterek kiíratása (kiir-char.c)

```
#include <stdio.h>
int main(){
  char c; /* vesd össze: int c; */
  for(c = 64; c < 96; c++) {
    printf(" %c", c);
  }
  putchar('\n');
  return 0;
}
```

Típussal kapcsolatos feladatok (char) II

Feladat: Írasd ki az 'a' és 'z' közé eső karakterek ASCII kódjait!

Kódok kiíratása (kiir-kod.c)

```
#include <stdio.h>
int main(){
  char c;
  for(c = 'a'; c <= 'z'; c++) {
    printf(" %hhd", c);
  }
  putchar('\n');
  return 0;
}
```

Megjegyzés: a `h` formátum karakter a méret felezésére utal, így egy `byte` méretű számot fogunk kiírni.

Típussal kapcsolatos feladatok (char) III

Feladat: Mi a különbség a signed char és az unsigned char értékkészlete között? Írasd ki -128-tól 255-ig egy signed és egy unsigned char típusú változó számértékét!

Típussal kapcsolatos feladatok (char) IV

signed és unsigned char különbsége (kiir-ertekkeszlet-char.c)

```
#include <stdio.h>
int main(){
  int i;
  signed char sc;
  unsigned char uc;
  for(i = -128; i <= 255; i++) {
    sc = i;
    uc = i;
    printf("%hhd %hhu\n", sc, uc);
  }
  return 0;
}
```

Házi Feladat: Az előző feladatban a *0-31* kódok kivételével írasd ki magát a karaktert is!

Típussal kapcsolatos feladatok (char) V

Feladat: Olvass be két legfeljebb 20 karakter hosszúságú szót, és fűzd őket egymás után egy harmadik sztringbe. A string.h függvényeit használd!

Sztringek összefűzése (osszefuz.c)

```
#include <stdio.h>
#include <string.h>
int main(){
  char egyik[21], masik[21], harmadik[41];
  scanf("%s %s", egyik, masik);
  strcpy(harmadik, egyik);
  strcat(harmadik, masik);
  printf(" -> %s\n", harmadik);
  return 0;
}
```

Típussal kapcsolatos feladatok (float/double) I

Feladat: Hány bájton tárolódik a float és double típus?

float és double méretek (meret-float.c)

```
#include <stdio.h>
int main(){
  printf("float : %d\n", sizeof(float));
  printf("double: %d\n", sizeof(double));
  return 0;
}
```

Feladat: Mi a különbség a float és a double pontossága között? Add hozzá az 1, 0.1, 0.01, 0.001, ... sorozat elemeit egy-egy *float* és *double* változóhoz. Milyen értékeket kapsz lépésenként?

Típussal kapcsolatos feladatok (float/double) II

float és double méretek (kiir-pontossag.c)

```
#include <stdio.h>
int main(){
  int i;
  float  f = 0.0, df = 1.0;
  double d = 0.0, dd = 1.0;
  for(i = 0; i < 20; i++) {
    f += df;
    d += dd;
    df *= 0.1;
    dd *= 0.1;
    printf("%d: float: %22.20f; double: %22.20lf\n", i, f, d);
  }
  return 0;
}
```

Típussal kapcsolatos feladatok (float/double) III

Feladat: Mi a különbség a float és a double értékkészlete között? Szorozgasd egy float és double változó értékét 0.1-del, amíg 0 nem lesz mindkettő! Milyen értékeket kapsz lépésenként?

float és double értékkészletek (kiir-ertekkeszlet-float.c)

```
#include <stdio.h>
int main(){
  float  f = 1.0;
  double d = 1.0;
  do {
    printf("float: %f; double: %lf\n", f, d);
    f *= 0.1;
    d *= 0.1;
  } while((f != 0.0) || (d != 0.0));
  return 0;
}
```

Típussal kapcsolatos feladatok (int) I

Feladat: Hány bájton tárolódnak a *short int*, *int*, *long int*, *long long* típusok?

int méretek (meret-int.c)

```
#include <stdio.h>
int main(){
  printf("short int: %zu\n", sizeof(short int));
  printf("int      : %zu\n", sizeof(int));
  printf("long int : %zu\n", sizeof(long int));
  printf("long long: %zu\n", sizeof(long long));
  return 0;
}
```

Típussal kapcsolatos feladatok (int) II

Feladat: Mi a különbség ugyanazon típus előjeles és előjeltelen verziója között? Deklarálj 6 változót (*signed/unsigned*, *short/long/long long*) változót, 0 kezdőértékkel, és vonj ki belőlük egyet! Milyen értékeket kapsz? Add értékül a változóknak a legnagyobb előjelesen ábrázolható értéket (ez fele az előjeltelen maximális értéknek), és adj hozzá egyet! Most mik a változók értékei? (kiir-ertekkeszlet-int.c)

```
#include <stdio.h>
int main(){
  signed short int ssi = 0;
  unsigned short int usi = 0;
  signed long int sli = 0;
  unsigned long int uli = 0;
  signed long long sll = 0;
  unsigned long long ull = 0;
  ssi -= 1;
```

Típussal kapcsolatos feladatok (int) III

```
usi -= 1;
sli -= 1;
uli -= 1;
sll -= 1;
ull -= 1;
printf("0 mínusz 1\n");
printf("s16: %hd\n", ssi);
printf("u16: %hu\n", usi);
printf("s32: %ld\n", sli);
printf("u32: %lu\n", uli);
printf("s64: %lld\n", sll);
printf("u64: %llu\n", ull);
ssi = usi /= 2;
sli = uli /= 2;
sll = ull /= 2;
printf("Legnagyobb ábrázolható előjeles szám...\n");
```

Típussal kapcsolatos feladatok (int) IV

```
printf("s16: %hd\n", ssi);
printf("u16: %hu\n", usi);
printf("s32: %ld\n", sli);
printf("u32: %lu\n", uli);
printf("s64: %lld\n", sll);
printf("u64: %llu\n", ull);
ssi += 1;
usi += 1;
sli += 1;
uli += 1;
sll += 1;
ull += 1;
printf("... plusz 1\n");
printf("s16: %hd\n", ssi);
printf("u16: %hu\n", usi);
printf("s32: %ld\n", sli);
```

Típussal kapcsolatos feladatok (int) V

```
  printf("u32: %lu\n", uli);
  printf("s64: %lld\n", sll);
  printf("u64: %llu\n", ull);
  return 0;
}
```

printf és scanf formátumok I

- A *printf* formátumok alakja mindig:
  `%[flag][width][.precision][length]type`
  Vagyis:
  - A % jellel kezdjük.
  - Lehetséges paraméterek: flags (igazítással kapcsolatos), width (szélesség), precision (pontosság), length (méret), type (típus).
  - A végén mindig jelezzük, hogy milyen típust használjunk (type).
  - A []-közötti paraméterek opcionálisak, azaz nem kötelező őket használni.
  - A paraméterek sorrendje kötött.

printf és scanf formátumok II

- A *scanf* formátumok alakja mindig: `%[*][width][modifiers]type`
  Vagyis:
  - A % jellel kezdjük.
  - Lehetséges paraméterek: * (input figyelmen kívül hagyása), width (szélesség), modifiers(pl. milyen méretű változóban tároljuk), type (típus).
  - A végén mindig jelezzük, hogy milyen típust használjunk (type).
  - A []-közötti paraméterek opcionálisak, azaz nem kötelező őket használni.
  - A paraméterek sorrendje kötött.
- Érdemes használni a `-Wall` fordítási kapcsolót.
- Leírások:
  - `man 3 printf`, `man 3 scanf`
  - Fritsi Dániel fordította összefoglaló

printf és scanf feladatok I

Feladat: Írj egy programot, ami beolvas egy *előjeltelen* `short int` értéket, és *nyolcas számrendszerbe* átváltva írja ki!

Oktális számrendszerben kiíratás (oktalis.c)

```
#include <stdio.h>
int main(){
  unsigned short int v;
  scanf("%hu", &v);
  printf("%ho\n", v);
  return 0;
}
```

Magyarázat: a beolvasásnál használjuk a felező *modifiert* (`h`, ettől lesz `short`), *előjeltelen* számra típusra (`u` type), a kiíratásnál használjuk ismét a felező *modifiert* (`h`), és az *oktális*(`o`) típust (type).

printf és scanf feladatok II

Feladat: Írj egy programot, ami beolvas egy hexadecimális egész számot, majd 15 karakter szélességben kiírja a decimális értékét, mindenképpen előjellel és vezető nullákkal!

Hexadecimális beolvasás (hexadecimalis.c)

```
#include <stdio.h>
int main(){
  unsigned int v;
  scanf("%x", &v);
  printf("%+015u\n", v);
  return 0;
}
```

printf és scanf feladatok III

Magyarázat: `x` jelenti a hexadecimálist, a kiíratásnál pedig + egy *flag*, ami megmondja, hogy mindig írjuk előjelet. A `0` jelenti, hogy nullákkal egészítsük ki a szám elejét, `15` pedig azt jelenti (*width*), hogy ilyen hosszú számot szeretnénk kapni.

printf és scanf feladatok IV

Feladat: Olvass be egy double és egy egész értéket, majd a valós értéket írasd ki az egészben megadott pontossággal!

Pontosság megadása (pontossag.c)

```
#include <stdio.h>
int main(){
  double ertek;
  int pontossag;
  scanf("%lf %d", &ertek, &pontossag);
  printf("%1.*lf\n", pontossag, ertek);
  return 0;
}
```

printf és scanf feladatok V

Magyarázat:

- Beolvasásnál:
  - `lf` a `double` típus, `d` az `int` típus jele.
- Kiíratásnál:
  - Az `1` megadja a *szélesség*et (minimum ennyi karakter).
  - A `.` után jönne a *pontosság*. Itt a `*` azt jelenti, hogy a pontosság számértékét a következő paraméter (jelen esetben az `ertek`) adja meg.
  - `lf` a `double` típus jele.

Házi Feladat: Mi a különbség a %g, %f, %e kiíratási formák között?

printf és scanf feladatok VI

Feladat: Olvass be egy csupa kisbetűből álló, legfeljebb 20 karakteres sztringet, majd írasd ki 10 karakteren jobbra igazítva az első legfeljebb 8 karakterét! A bemeneten a kisbetűket közvetlenül bármi követheti.

Sztringek kezelése (sztringek.c)

```
#include <stdio.h>
int main(){
  char str[21];
  scanf("%20[a-z]", str);
  printf("%10.8s\n", str);
  return 0;
}
```

printf és scanf feladatok VII

Magyarázat:

- A 20 karakter hosszú sztringet 21 méretű tömbben tároljuk (utolsó helyen a lezáró `'\0'`).
- Beolvasásnál:
  - Megadjuk a méretet: 20
  - Megadjuk a karakterek lehetséges halmazát: `[a-z]` (ez hasonló a bash-nél látottakhoz). Ha olyan karakterhez érünk, amelyik nem eleme a halmaznak, akkor befejezzük a beolvasást.
- Kiíratásnál:
  - Alapértelmezésben jobbra igazítottan írunk ki (a `-` flag jelenti a balra igazítást).
  - Megadjuk, hogy hány karaktert akarunk kiíratni: 10
  - A `.` után jelezzük a pontosságot, jelen esetben azt, hogy a sztring első hány karakterét akarjuk kiíratni: 8

printf és scanf feladatok VIII

Feladat: Egy sor kiíratási formátuma: `"nev: %s; pont: %d;"`. Olvasd be a kiírt számot úgy, ha tudod, hogy a kiírt sztring nem tartalmazhat pontosvesszőt! Ellenőrízd le, hogy az input sor valóban helyes-eű!

Sztringek kezelése (sztringek2.c)

```
#include <stdio.h>
int main() {
  int val, ret;
  ret=scanf("nev: %*[^;]; pont: %d;", &val);
  if(ret == 1) {
    printf("A szam: %d\n", val);
  } else {
    printf("Helytelen formatum\n");
  }
  return 0;
}
```

printf és scanf feladatok IX

Magyarázat:

- A *scanf* visszatérési értéke megmondja, hogy hány változónak adott sikeresen értéket. Jelen esetben ez 1, ha a formátum megfelelő volt, hiszen az első sztringet figyelmen kívül hagyjuk (*).
- Az első sztring nem tartalmazhat pontosvesszőt, ezt ellenőriznünk kell azzal, hogy a `[^;]` korlátozást megadjuk (^ a komplementer halmazt jeleneti).

printf és scanf feladatok X

Feladat: Írasd ki a fájlvége jelig (`^D`) tartó bemenetet úgy, hogy a számjegyeket törlöd belőle. A végén írd ki, hogy hány számjegyet töröltél.

printf és scanf feladatok XI

Számjegyek figyelmen kívül hagyása (szamjegyek.c)

```
#include <stdio.h>
int main() {
  int c, d = 0;
  while((c = getchar()) != EOF) {
    if('0' <= c && c <= '9') {
      d++;
    } else {
      putchar(c);
    }
  }
  printf("\n--\n%d darab torolve\n", d);
  return 0;
}
```

printf és scanf feladatok XII

Magyarázat:

- `EOF` jelenti a `^D`-t (-1-es `signed char` érték).
- `getchar()` beolvas egy karaktert `stdin`-ről.

Házi Feladat: Olvass be egy teljes sort egy sztringbe, majd írasd ki. Mi a különbség a `gets`, `puts` illetve `fgets(stdin, ...)`, `fputs(stdout, ...)` használata között?

FILE I/O I

```
#include <stdio.h>

FILE *fopen(const char *path, const char *mode);
int fclose(FILE *fp);

int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

fopen A futtató számítógép fájlrendszerén levő fájl megnyitása. A fájl (relatív vagy abszolút) útvonalát meg kell adni a `path` paraméterben. A `mode` szöveges paraméter megmondja, hogy milyen módon nyissuk meg a fájlt. A következő karakterekből állhat:

- r csak olvasásra, a mutató a fájl elején
- r+ olvasásra és írásra, a mutató a fájl elején

FILE I/O II

- **w** csak írásra; a fájl tartalmának törlése; a mutató a fájl elején
- **w+** olvasásra és írásra; ha a fájl nem létezik, létrejön; ha létezik, tartalma törlődik; a mutató a fájl eljén
- **a** hozzáfűzés (írás a fájl végére); a fájl létrejön, ha nem létezett; a mutató a fájl végén
- **a+** olvasás és hozzáfűzés (írás a fájl végére); a fájl létrejön, ha nem létezett; a kezdeti olvasó mutató a fájl elején, az írás mindig a fájl végére történik

A függvény sikertelen esetben NULL-t ad vissza, sikeres esetben pedig egy FILE mutatót.

FILE I/O III

fclose Egy megnyitott fájl lezárása. Mindig zárjunk le minden fájlt, amit megnyitottunk. Siker esetén 0-t, sikertelen esetben `EOF`-t ad vissza és feltöltődik az `errno` változó a hiba típusával.

fprintf, fscanf Teljesen hasonlóan működnek a `printf` és `scanf` függvényekhez, azonban fájlba írnak és fájlból olvasnak. Az első paraméter a megnyitott fájl azonosítója.

Bővebb leírás:

- man 3 fopen
- man 3 fclose
- man 3 fprintf
- man 3 fscanf

Példa fájl I/O-ra I

Ez a példa bemutatja, hogyan olvassunk és írjuk fájlokat. A példa program beolvas két számot a `be.txt`-ből (ha olvasható) és kiszámítja a két szám összegét és szorzatát, az eredményt a `ki.txt`-be írja (ha írható).
A fájlkezelés elengedhetetlen a kötelező programok megoldásához. Példa program (file-io.c):

```
#include <stdio.h>

int main() {
  int a, b;
  FILE *infile;
  FILE *outfile;
  if(!(infile = fopen("be.txt", "r"))) {
        return 1;
  }
  if(!(outfile = fopen("ki.txt", "w"))) {
```

Példa fájl I/O-ra II

```
        fclose(infile);
        return 1;
    }

    fscanf(infile, "%d %d", &a, &b);
    fprintf(outfile, "Osszeg: %d\nSzorzat: %d\n", a+b, a*b);

    fclose(infile);
    fclose(outfile);
    return 0;
}
```

C pointerek I

- A C nyelvben vannak pointerek (mutatók), amelyek igen hasznosak és közvetlen memória hozzáférést biztosítanak. Egyes dolgokat nem is tudnánk megcsinálni a pointer típus nélkül.
- Egy változót pointer tipusúra deklarálhatunk, ha a deklaráció során a * jelet tesszük a típus és a változónév közé.
- Pl. az `int *a;` vagy `int* a;` deklarációk egy `int` típusú *pointer*t hoznak létre. Látni fogjuk, hogy az előbbi kedvezőbb bizonyos esetekben.
- A pointerünk mutathat egy változó memóriaterületére, vagy egy általunk megadott memóriaterületre is.
- Egy változó memóriacímét megkaphatjuk a & jellel, ezt használtuk pl. `scanf` esetében.

C pointerek II

- A pointerrel a memóriaterületre írhatunk vagy olvashatunk (ha az írható illetve olvasható). A pointer által mutatott memóriaterület tartalmát megkaphatjuk, ha a pointer neve elé egy *-ot teszünk, azaz pl. `*a` megadja az a memóriacímen található `sizeof(int)` bájtot foglaló értéket, az `*a = 5;` utasítással erre a memóriaterületre 5-ös értéket írhatunk (`int` szélességben).
- A pointer egy adott területét éri el a memóriának, ezt pont az a terület, ahova mutat. A terület mérete a pointer típusától függ. Pl. egy `int*` típus `sizeof(int)` méretű memóriaterületet ér el.
- A pointereket lehet egész számoknak (`void*`) tekinteni, amelyekhez hozzáadhatunk, kivonhatunk, stb. Ez azt befolyásolja, hogy a memória mely területére mutat a pointer (relatíve).

C pointerek III

Pl. vegyük a következő kódrészletet:

```
int x = 25;
short int *p = x;
printf("%hd %hd\n", *p, *(p+1));
```

Deklarálunk egy `int` típusú `x` változót, és egy rá mutató `short int` típusú pointert. Tegyük fel, hogy `sizeof(int) == 4` és `sizeof(short int) == 2`. Ekkor `p` az `i` változó alsó 2 bájtjára mutat, `p+1` pedig a felső két bájtjára. Gondoljuk meg, hogy a program mit ír ki kis endián illetve nagy endián gépen!

- A pointerünket léptethetjük is, ekkor a pointer egész pontosan a típusa méretével azonos távolságot lép. Pl. ha egy `int*`-ot mozgatunk eggyel (pl. növeljük 1-el), akkor az 1*sizeof(int)-el előrébb található területre fog mutatni. Ez teljesen ugyan az, mintha egy tömbben mozognánk az elemek felett, a tömb változó valójában a tömb 0. elemére mutató pointer típus.

C dinamikus memória kezelés I

- A dinamikus memória lényege, hogy olyan mennyiségű memóriára lenne szükségünk, amit fordítási időig még nem tudunk eldönteni (pl. bemenettől függ), és nem szeretnénk feleslegesen lefoglalni a maximális területet előre.
- Futás közben létrehozhatunk (*allokál*hatunk) egy dinamikus memóriaterületet, ez a *heap*-en fog létrejönni, és nem pl. a *stack*-en.
- Ha allokáltunk egy memóriaterületet, akkor azon dolgozhatunk, és ha befejeztük a munkát rajta, érdemes felszabadítani (*deallokál*ni).
- Azt a jelenséget, amikor a program(ozó) nem szabadít fel egy memóriaterületet, és nem is használja, esetleg nem is tud róla, *memóriaszivárgás*nak (*memory leak*-nek) nevezzük. Ez nem kívánatos, hiszen addig más nem tudja használni a memóriaterületet.
- Egy kiváló eszköz a memory leakek megtalálására a `valgrind` nevű program, amely képes egyéb hibákat is felderíteni.

C dinamikus memória kezelés II

```
#include <stdlib.h>

void *calloc(size_t nmemb, size_t size);
void *malloc(size_t size);
void free(void *ptr);
void *realloc(void *ptr, size_t size);
```

- A `malloc` megpróbál egy `size` méretű memóriaterületet allokálni, ha sikerrel jár, visszatér a memóriaterület címével; ha nem jár sikerrel, `NULL`-t (azaz `(void*) 0`-t) ad vissza.
- A `calloc` hasonlóan működik, azonban `nmemb*size` méretű memóriaterületet foglal, és a memóriát feltölni 0 bitekkel.

C dinamikus memória kezelés III

- A `free` szolgál memóriaterület felszabadítására. Paraméterül természetesen egy mutatót kap a felszabadítandó memóriaterületre. Ha `ptr == NULL`, nem történik semmi. Nem dinamikusan allokált címet ne adjunk paraméterül. Többször ne próbáljuk meg felszabadítani ugyan azt a területet (hacsak nem allokáltuk újra).
- A `realloc` a `malloc`-hoz hasonlítható (`ptr == NULL` esetén teljesen azonos). A paraméterül kapott memóriaterületet megpróbálja megnövelni (ha kell, átmozgatja, de a tartalmát nem változtatja), ha sikerrel jár, visszatér a memóriaterület címével; ha nem jár sikerrel, `NULL`-t (azaz `(void*) 0`-t) ad vissza.

C dinamikus memória, pointer feladatok I

Példa: Nézzük meg, mi a különbség p, q, illetve *p és *q értéke között! (pointerek.c)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  int *p, *q;
  p = malloc(sizeof(int));
  q = malloc(sizeof(int));
  *p = 3;
  *q = 3;
  printf("p es q %s\n", p == q ? "megegyezik" :
                                 "nem egyezik meg");
  printf("*p == %d, *q == %d\n", *p, *q);
  *p = 4;
  printf("*p == %d, *q == %d\n", *p, *q);
```

C dinamikus memória, pointer feladatok II

```
  free(p);
  p = q;
  printf("p es q %s\n", p == q ? "megegyezik" :
                                 "nem egyezik meg");
  printf("*p == %d, *q == %d\n", *p, *q);
  *p = 4;
  printf("*p == %d, *q == %d\n", *p, *q);
  free(p);
  return 0;
}
```

Magyarázat:

- Két mutatónk lesz, amelyek dinamikusan foglalt `sizeof(int)` bájt hosszú memóriaterületre mutatnak.
- Mindkét memóriablokkot feltöltjük a 3 `sizeof(int)` hosszúságú értékkel.

C dinamikus memória, pointer feladatok III

- Kiíratjuk a két memóriaterület értékét.
- Megnézzük, hogy `p == q` teljesül-e. Természetesen nem teljesül, hiszen a két változónkat külön helyen tároljuk a memóriában (a két memóriacím nem egyenlő).
- A `p` mutató által mutatott területet feltöltjük 4-gyel, és ezt kiíratással ellenőrizzük.
- A `p` mutató ezentúl mutasson ugyan oda, ahova `q` mutat. Nagyon fontos, hogy ezelőtt felszabadítjuk a `p` korábbi memóriaterületét, hiszen ezen a ponton túl már nem tudjuk megmondani, hogy mi volt a `p` terület címe korábban.
- Látjuk, hogy ekkor már megegyezik a két mutató, hiszen ugyan arra a memóriacímre mutatnak.
- Ebből kifolyólag ugyan arra az `int` értékre is hivatkoznak.
- Ne felejtsünk el megint felszabadítani.

C dinamikus memória, pointer feladatok IV

Példa: Futtassuk le a következő programot, és értelmezzük! Melyik érték melyik értékkel egyenlő, és miért ?

Pointerek példa (cim.c)

```
#include <stdio.h>
int main(){
  int a = 10;
  int *pa;
  pa = &a;
  printf("%d %#x\n", a, (int)pa);
  printf("%#x %#x\n", (int)&a, (int)&pa);
  printf("%d\n", *pa);
  return 0;
}
```

C dinamikus memória, pointer feladatok V

Magyarázat:

- Deklarálunk egy (statikus) `int` típusú változót, a 10 értékkel.
- Deklarálunk egy (statikus) `int*` típusú változót (`int`-re mutató *pointer*), és az `a` változó memóriacímét (referencia jel: `&`), adjuk neki értékül.
- Ezek után kiíratjuk az `a` változó értékét, a `pa` mutató memóriacímét, az `a` változó memóriacímét, a `pa` memóriacímét, és a `pa` által mutatott memóriaterületet egészként. A memóriacímeket hexadecimális alakban írjuk ki.
- Mint minden változónak, a pointer változóknak is van memóriacímük (azaz a memóriában hol tároljuk egy a pointer számbeli értékét, azaz, hogy hova mutat).

C dinamikus memória, pointer feladatok VI

Feladat: Írj egy `csere(int x, int y)` függvényt, ami megcseréli két `int` típusú változó értékét! (csere.c)

```
#include <stdio.h>
void csere(int x, int y){
  int tmp;
  tmp = x;
  x   = y;
  y   = tmp;
}
int main(){
  int x = 3, y = 4;
  printf("A fuggveny elott: x = %d, y = %d\n", x, y);
  csere(x,y);
  printf("A fuggveny utan: x = %d, y = %d\n", x, y);
  return 0;
```

C dinamikus memória, pointer feladatok VII

```
}
```

Ez így hibás, mert csak a lokális változókat cseréli, a hatás a függvényblokkon kívül nem érvényesül.

C dinamikus memória, pointer feladatok VIII

Feladat: Javítsuk az előző programot! (csere2.c)

```
#include <stdio.h>
void csere(int *x, int *y){
  int tmp;
  tmp = *x;
  *x  = *y;
  *y  = tmp;
}
int main(){
  int x = 3, y = 4;
  printf("A fuggveny elott: x = %d, y = %d\n", x, y);
  csere(&x,&y);
  printf("A fuggveny utan: x = %d, y = %d\n", x, y);
  return 0;
}
```

C dinamikus memória, pointer feladatok IX

Feladat: Deklarálj egy 20 elemű `int` tömböt, majd töltsd fel értékekkel az inputról. Deklarálj egy pointert, és a beolvasást azon keresztül valósítsd meg. (tombfeltolt.c)

```
#include <stdio.h>
#define N 20
int main(){
  int t[N], *p, i;
  for(i = 0; i < N; i++) {
    p=&(t[i]);
    scanf("%d", p);
  }
  for(i = 0; i < N; i++) {
    printf("%d\n", t[i]);
  }
  return 0;
```

C dinamikus memória, pointer feladatok X

```
}
```

C dinamikus memória, pointer feladatok XI

Magyarázat:

- Deklarálunk (statikusan) egy N elemszámú `int` típusú tömböt, egy `int*`-ot, (itt látható, hogy a változó elé érdemes rakni a *-ot és nem a típus után), és egy `i` ciklusváltozót.
- A `p` változó a ciklus aktuális (`i`-edik) elemére mutat, és 0-tól $N-1$-ig beolvasunk az i. eleme memóriaterületére.
- Kiíratjuk a tömb elemeit.

C dinamikus memória, pointer feladatok XII

Feladat: Az előzőhöz hasonló a feladat, csak most először olvasd be a tömb méretét, és foglalj neki dinamikusan helyet! (dintomb.c)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  int *t, *p, i, N;
  scanf("%d", &N);
  t=(int*)malloc(N*sizeof(int));
  for(i = 0; i < N; i++) {
    p=&(t[i]);
    scanf("%d", p);
  }
  for(i = 0; i < N; i++) {
    printf("%d\n", t[i]);
  }
```

C dinamikus memória, pointer feladatok XIII

```
  free(t);
  return 0;
}
```

Házi feladatok:

1. A `malloc` helyett használd a `calloc` függvényt!
2. A tömb elemeit a `p` pointer `i`-vel növelésével érd el!
3. A tömb elemeit úgy érd el, hogy lerögzítesz egy a tömb végére mutató pointert, és egyet, amely az elejére mutat, majd az elejére mutatót addig növeled, míg nem egyenlő a végére mutatóval!

C dinamikus memória, pointer feladatok XIV

Feladat: Olvass be 5 darab maximum 99 karakter hosszú szót úgy, hogy mindegyiknek pontosan annyi helyet foglalsz, amennyi kell! A sztringeket írasd ki, majd szabadítsd fel a lefoglalt területet! (karaktertomb.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(){
  char buff[100];
  char *ptr_tomb[5];
  int i;
  for(i = 0; i < 5; i++) {
    scanf("%s", buff);
    ptr_tomb[i] = (char*)malloc(strlen(buff)+1);
    strcpy(ptr_tomb[i], buff);
  }
```

C dinamikus memória, pointer feladatok XV

```
  for(i = 0; i < 5; i++) {
    puts(ptr_tomb[i]);
  }
  for(i = 0; i < 5; i++) {
    free(ptr_tomb[i]);
  }
  return 0;
}
```

Feladatok

További feladatok a `/pub/progalap/Gyakorlat/gyak08/` alatt.

C struct I

- A `struct` egy több típusból összeálló összetett adatszerkezet.
- A struktúrának adattagjai vannak, amelyek különböző típusúak lehetnek.
- A tömbökhöz hasonló megoldás, de itt a tagoknak nem indexük, hanem nevük van.
- Az adattagok a memóriában egymás után folytonosan tárolódnak, kezdetben inicializálatlanok.
- Példa:

```
typedef struct s_eredmeny {
  char nev[20];
  int pontszam;
  char *megoldas;
} eredmeny;
// sizeof(eredmeny) == 28 (x86-on)
```

C struct II

- A struktúra teljes méretét lekérdezhetjük a `sizeof()` operátorral. Vigyázat, a fordító alkalmazhat igazítást, így a struktúra mérete nagyobb lehet, mint az adattagok mérete összesen!
- A tagokra a `.` operátorral hivatkozunk. Ha struktúra pointerről van szó, akkor a `->` operátort kell használnunk:

```
eredmeny a;
eredmeny* pa = &a;
a.pontszam = 20;
a->nev = "Ab Cd";
```

- Struktúrák inicializálása:

```
eredmeny a = { "Nev", 1234, "Megoldas: ..." };
```

vagy

```
eredmeny a = { .nev = "Nev", .pontszam = 1234,
               .megoldas = "Megoldas: ..." };
```

C struct III

- A struktúra nem lehet rekurzív, azonban tartalmazhat saját típusú pointert:

```
typedef struct csucs {
  int adat;
  struct csucs* bal;
  struct csucs* jobb;
} pont;
```

C union I

- A `union` a `struct`-hoz képest annyiban különbözik, hogy a tagok memóriaterületei átfedik egymást.
- A felhasznált memóriaterület méret az egyes tagok méretének maximuma.
- A dolog lényege, hogy egy adatterületet többféleképpen is felhasználhatunk.
- Példa:

```
typedef union u_szam {
  int egesz;
  short fele[2];
} u;
// sizeof(u) == 4 (x86-on)
```

(azaz könnyedén elérhetjük egy 4-bájtos szám alsó és felső 2 bájtját)

- Egy tag elérése hasonló a `struct`-nál látottakhoz.

C union II

- Uniók inicializálása:

```
u c = { .egesz = 3 };
u d = { 3 };
```

C struct és union feladatok I

Feladat: Hozz létre típust egy háromdimenziós térbeli pozíció tárolására. Ezt felhasználva hozz létre egy típust, ami részecskék helyzetét, tömegét, nevét és töltését (pozitív/negatív/semleges) tárolja. Készíts egy függvényt, ami két részecskéről eldönti, hogy melyik nehezebb, és egy másikat, ami megmondja, hogy elektromosan vonzzák vagy taszítják egymást, esetleg nem hatnak egymásra. Inicializálj két részecskét, és használd a függvényeket. (`struct.c`)

```
#include <stdio.h>
typedef struct {
  double x, y, z;
} pozicio;

typedef char nevtipus[30];
typedef enum {negativ = -1, semleges, pozitiv} toltestipus;
typedef struct {
```

C struct és union feladatok II

```
  pozicio helyzet;
  double tomeg;
  nevtipus nev;
  toltestipus toltes;
} reszecske;

int tomeghasonlitas(reszecske a, reszecske b){
  if(a.tomeg < b.tomeg) {
    return -1;
  }
  if(a.tomeg > b.tomeg) {
    return 1;
  }
  return 0;
}
int vonzas(reszecske a, reszecske b){
```

C struct és union feladatok III

```
  if(a.toltes == semleges || b.toltes == semleges) {
    return 0;
  }
  return (a.toltes == b.toltes) ? 1 : -1;
}
int main(){
  reszecske p={{0.0, 0.0, 0.0}, 1.0,
    "proton", pozitiv};
  reszecske e={{1.0, 1.0, 1.0}, 0.001,
    "elektron", negativ};
  printf("tomeg: %d\nvonzas: %d\n",
    tomeghasonlitas(p, e), vonzas(p, e));
  return 0;
}
```

C struct és union feladatok IV

Feladat: Adott a síkon 3 pont, mi az általuk meghatározott háromszög területe? (haromszog.c)

```
#include <stdio.h>
#include <math.h>
struct pont {
  float x;
  float y;
};
float tav(struct pont P, struct pont Q) {
  return sqrtf((P.x - Q.x) * (P.x - Q.x) +
               (P.y - Q.y) * (P.y - Q.y));
}
int main() {
  struct pont A, B, C;
  float a, b, c, s;
```

C struct és union feladatok V

```
  scanf("%f %f", &A.x, &A.y);
  scanf("%f %f", &B.x, &B.y);
  scanf("%f %f", &C.x, &C.y);
  a = tav(B, C);
  b = tav(A, C);
  c = tav(A, B);
  s = (a + b + c) / 2;
  printf("Terulet: %f\n", sqrtf(s * (s - a) * (s - b) *
                                (s - c)));
}
```

Mivel használjuk a `math.h`-t, a fordításnál használnunk kell a `-lm` kapcsolót.

C struct és union feladatok VI

Feladat: Készítsünk komplex számok tárolására alkalmas adatszerkezetet (egész komponensekkel). Készítsünk továbbá olyan függvényeket, melyek feladata (`komplex.c`):

- kiír egy komplex számot az stdout-ra,
- összead két komplex számot, és visszaadja az eredményt
- összeszoroz két komplex számot, és visszaadja az eredményt

```
#include <stdio.h>
typedef struct komplex {
  int real;
  int imag;
} komplex;
komplex add(komplex k1, komplex k2){
  komplex e;
  e.real = k1.real + k2.real;
```

C struct és union feladatok VII

```
  e.imag = k1.imag + k2.imag;
  return e;
}
komplex mul(komplex k1, komplex k2){
  komplex e;
  e.real = k1.real * k2.real - k1.imag * k2.imag;
  e.imag = k1.imag * k2.real + k1.real * k2.imag;
  return e;
}
void printk(komplex k){
  printf("(%d%+di)\n", k.real, k.imag);
}
int main(){
  komplex x1,x2,e;
  x1.real = 10;
  x1.imag = 2;
```

C struct és union feladatok VIII

```
  x2.real = 20;
  x2.imag = -3;
  printk(x1);
  printk(x2);
  e = add(x1,x2);
  printk(e);
  printk(mul(x1,x2));
  return 0;
}
```

C struct és union feladatok IX

Feladat: Láncolt lista. Olvassunk be egész számokat egy láncolt listába egy adott végjelig, majd írassuk ki őket! (`linkedlist.c`)

```
#include <stdio.h>
#include <stdlib.h>
#define VEGJEL 0
struct cella {
  int ertek;
  struct cella *kov;
};
int main(){
  struct cella *elso = NULL;
  struct cella *p;
  int i;
  scanf("%d", &i);
  while(i != VEGJEL) {
```

C struct és union feladatok X

```
    p  = (struct cella*)malloc(sizeof(struct cella));
    p->ertek = i;
    p->kov   = elso;
    elso     = p;
    scanf("%d", &i);
  }
  for(p = elso; p != NULL; p = p->kov) {
    printf("%d\n", p->ertek);
  }
  while(elso != NULL) {
    p    = elso;
    elso = p->kov;
    free(p);
  }
  return 0;
}
```

C struct és union feladatok XI

Házi Feladat: A kiíratáskor írasd ki a cella címét, a cella két mezőjének címét és értékét is. Hasonlítsd össze a cellák címeit a kov mezők értékeivel!

C struct és union feladatok XII

Feladat: Mi a különbség a struct és a union között? Deklarálj egy struct és egy union típust ugyanolyan mezőkkel. Adj értéket a mezőknek, majd írasd ki őket! (union.c)

```
#include <stdio.h>
typedef struct {int i; double d; char c; float f;} st;
typedef union {int i; double d; char c; float f;} un;
int main(){
  st s;
  un u;
  s.i = u.i = 12345;
  s.d = u.d = 3.141593;
  s.c = u.c = 'A';
  s.f = u.f = 2.718281;
  printf("s.i: %d  u.i: %d\n", s.i, u.i);
  printf("s.d: %lf  u.d: %lf\n", s.d, u.d);
```

C struct és union feladatok XIII

```
  printf("s.c: %c  u.c: %c\n", s.c, u.c);
  printf("s.f: %f  u.f: %f\n", s.f, u.f);
  return 0;
}
```

C struct és union feladatok XIV

Feladat: Írasd ki a mezők memóriacímét! (union-cim.c)

```
#include <stdio.h>
typedef struct {int i; double d; char c; float f;} st;
typedef union {int i; double d; char c; float f;} un;
int main(){
  st s;
  un u;
  printf("s.i: %p  u.i: %p\n", &s.i, &u.i);
  printf("s.d: %p  u.d: %p\n", &s.d, &u.d);
  printf("s.c: %p  u.c: %p\n", &s.c, &u.c);
  printf("s.f: %p  u.f: %p\n", &s.f, &u.f);
  return 0;
}
```

C függvények - gyakorló feladatok I

Feladat (kimenő paraméterek): Készítsünk egy C programot, amely bemutatja a másodfokú egyenlet megoldását! (`masodfok.c`, fordításhoz kell itt a `-lm` kapcsoló)

C függvények - gyakorló feladatok II

Feladat (rekurzió): Állítsuk elő a Fibonacci sorozat *n*. elemét! (`fiborek.c`)

```
#include <stdio.h>
int fib(int n) {
  if(n == 1 || n == 2) {
    return 1;
  } else {
    return fib(n-1) + fib(n-2);
  }
}
int main() {
  int n;
  printf("n erteke?:\t");
  scanf("%d", &n);
  printf("A Fibonacci-sorozat %d. eleme:\t%d", n, fib(n));
```

C függvények - gyakorló feladatok III

```
  return 0; }
```

Házi Feladat: Vizsgáld meg, hogy hányszor hívódik meg az előbbi rekurzív függvény!

C pointerek - gyakorló feladatok I

Feladat: Olvasd be egy tömb méretét, foglalj neki dinamikusan helyet, majd olvasd be az elemeit! (`tomb.c`)

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  int *t, *p, i, N;
  scanf("%d", &N);
  t=(int*)malloc(N*sizeof(int));
  for(i = 0, p = t; i < N; i++, p++) {
    scanf("%d", p);
  }
  for(i = 0, p = t; i < N; i++) {
    printf("%d\n", *(p++));
  }
  free(t);
```

C pointerek - gyakorló feladatok II

```
  return 0;
}
```

C pointerek - gyakorló feladatok III

Feladat: Adott egy kétdimenziós tömb. Pointer segítségével járjuk be az összes elemét! (pointer2d.c)

```
#include <stdio.h>
#define SIZE 3
int main(){
  int tomb[SIZE][SIZE] =
    {{0, 1, 2 },
    {3, 4, 5 },
    {6, 7, 8 } };
  int i,j;
  int *pa = NULL;

  pa = (int*) tomb;  /* pa = &tomb[0][0] */
  for(i = 0; i< SIZE * SIZE; i++)
    printf("%2d ", pa[i]);
```

C pointerek - gyakorló feladatok IV

```
  printf("\n");
  for(i = 0; i< SIZE * SIZE; i++)
    printf("%2d ", *(pa+i));
  printf("\n");
  for(i = 0; i< SIZE * SIZE; i++, pa++)
    printf("%2d ", *pa);
  printf("\n");

  /* vigyázat! mivel pa-t növeltük
   * a for ciklusban, ezért a ciklus után már
   * nem a tömb legelső elemére fog mutatni!
   */
  return 0;
}
```

C pointerek - gyakorló feladatok V

Feladat: Dinamikus kétdimenziós tömb létrehozása. (`dinpointer2d.c`)

```
#include <stdlib.h>
#include <stdio.h>
int main(){
    int *p, **t;
    int N = 3, M = 4;
    int i, j, v = 0;
    /* V1: egydimenziós tömb */
    p = malloc(N * M * sizeof(int));
    for(i = 0; i < N; i++) {
        for(j = 0; j < M; j++) {
            p[i * N + j] = v++;
        }
    }
    free(p);
```

C pointerek - gyakorló feladatok VI

```
    /* V2: sorokat külön-külön */
    t = malloc(N * sizeof(int*));
    for(i = 0; i < N; i++) {
        t[i] = malloc(M * sizeof(int));
    }
    for(i = 0; i < N; i++) {
        for(j = 0; j < M; j++) {
            t[i][j] = v++;
        }
    }
    for(i = 0; i < N; i++) {
        free(t[i]);
    }
    free(t);
    return 0;
}
```

C Tárolási osztályok

auto alapértelmezett, azt jelenti, hogy a memóriaterület ideiglenesen használt, ha kilépünk a blokkból, nem használjuk tovább

const konstans memóriaterület; csak fordítási időben ellenőrizhető, így pl. egy pointerrel változtathatunk az értékén.

register a fordító megpróbálja a változót egy külön regiszterben tárolni

volatile jelezzük a fordítónak, hogy a változó értéke egy külső hatás miatt módosulhat (pl. meghajtóprogram)

static a globális változók alapértelmezett tárolási osztálya; ha egy függvényblokkban használjuk, azt jelenti, hogy a függvény minden meghívásakor ugyan azt a memóriaterületet használjuk, a változót nem inicializáljuk újra és újra

extern a változónak később adunk kezdőértéket; több `.o` tárgykódból összeálló programok esetén van jelentősége

C Tárolási osztály feladatok

- Feladat: Készíts egy pointert, ami egy konstans értékre mutat!
- Feladat: Készíts egy konstans pointert, ami egy nem konstans értékre mutat!
- Feladat: Készíts egy pointert, ami egy tömbre mutat!

Megoldás (`dekl.c`):

```
#include <stdio.h>
int main(){
  const int *p=NULL;
  int * const c=NULL;
  int (*t)[20];

  p = malloc(sizeof(int));
  *p = 2007; /* HIBÁS */
  free(p);
  c = malloc(sizeof(int)); /* HIBÁS */
  *c = 2007;
```

C függvény pointerek I

- A C nyelvben van lehetőségünk függvény pointer létrehozására is.
- A szintaxis elsőre talán szokatlan lehet.
- Példa:

```
double fgv(double, double);
/* függvény deklarációja */
double (*fptr)(double, double);
/* ilyen típusú függvényre mutató pointer deklarációja */
fptr = fgv;
/* a függvény nevét adom kezdo"értékül,
 * a fordító persze ebbo"l címet állít elo" */
fptr(x, y);
/* meghívom a függvényt a pointeren keresztül */
```

Feladat: Tekintsük meg a függvény pointereket bemutató példa C programokat! (`fx.c`, `bejar.c`)

Feladatok

További feladatok a `/pub/progalap/Gyakorlat/gyak09/` alatt.

Parancssori paraméterek I

- Minden program képes parancssori paramétereket fogadni.
- Egy C program belépési pontjában (pl. `main`) a következő érhetőek el:
  - a parancssori paraméterek száma (`int argc`)
  - a parancssori paraméterek sztring tömbje (`char** argv` vagy `char* argv[]`)
  - a környezeti változók tömbje (`char** envp` ahol az utolsó elemet a `NULL` jelzi)
- Azaz: `int main(int argc, char** argv, char** envp){...}`
- Az `argv` 0. eleme mindig a futtatott program neve (ahogyan azt futtattuk).
  Ez teszi lehetővé pl. azt, hogy egy program másképp viselkedjen, ha más néven hívjuk meg.
  pl. `mc-mcedit`

Parancssori paraméterek feladatok I

Feladat: Írj egy programot, ami összeadja a parancssori paramétereit! (args.c)

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char* argv[]){
  int i;
  int arg = 0;
  printf("Összesen %d programargumentumot kaptam!\n", argc);
  for(i = 0; i< argc; i++)
    printf("%d : %s\n",i, argv[i]);
  if(argc > 1){
    for(i = 1; i < argc; i++)
      arg += atoi(argv[i]);
  }
  printf("Az argumentumok összege : %d\n", arg);
```

Parancssori paraméterek feladatok II

```
  return 0;
}
```

Feladat: Írj egy programot, amely n-szer egymás után fűzi ugyanazt az s sztringet, ahol n és s is parancssori paraméter!

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char* argv[]){
  if(argc < 3){
    fprintf(stderr, "Használat: %s <valami szám>", argv[0]);
    fprintf(stderr, " <valami sztring>\n");
    return 1;
  }
  int i;
```

Parancssori paraméterek feladatok III

```
  char *er = (char*) calloc (
                     atoi(argv[1]) + 1,
                     strlen(argv[2]));
  strcpy(er, argv[2]);
  for(i = 0;i < atoi(argv[1]) - 1; i++)
    strcat(er, argv[2]);
  printf("%s\n", er);
  free(er);
  return 0;
}
```

C makrók I

- A C nyelvben a `#define` utasítással deklarálhatunk egy makrót, amelyet a preprocesszor az így deklarált konstansokhoz hasonlóan be fog helyettesíteni a programkódunkba.
- A makrónknak lehetnek paraméterei.
- Hasznos makrókat tudunk létrehozni (`min.`, `max.`, `abs.`, `elemszám`, `debug`)
- Érdemes a makrónkat zárójelek közé tenni, ezzel elkerülve a prioritási sorrendből fakadó gondokat.
- Ha egy paraméterünk többször szerepel a makró definíciójában, akkor az többször is értékelődik ki (a preprocesszor csak behelyettesít!)
- Makrók hibás használatára példák:
  - Példa 1
  - Példa 2
  - Példa 3
  - Példa 4

C makró feladatok I

Példa makró (`makrofgv.c`):

```
#include <stdio.h>
#include <math.h>
#define min(X,Y)  ((X)<(Y)?(X):(Y))
int main() {
  printf("MIN(e^3, 3^2):\t%f\n", min( exp(3), pow(3, 2) ) );
  return 0;
}
```

Kérdés: Miért lehet hibás a `#define negyzet(X) X*X` makró?

C makró feladatok II

Másik példa (makrofgv2.c):

```
#include <stdio.h>
#define MAX(a,b) (((a) > (b) ) ? (a) : (b))
#define MIN(a,b) (((a) > (b) ) ? (b) : (a))
#define MIN3(a,b,c) (((a) < (b)) ? (((a) < (c)) ? (a) : (c)) :
/*
 * MINDENT zárójelezni kell, ugyanis ha így írnánk:
 * #define MAX(a,b) a > b ? a : b
 *
 * akkor MIN( a-3 , a?1:3 ) esetén az eredmény:
 * a-3 > a?1:3 ? a-3 : a?1:3 , ami nem az, amit szeretnénk!
 *
 */
int main(){
int a, b, c;
```

C makró feladatok III

```
  a = -23;
  b = 44;
  c = 0;
  printf("MAX(%d,%d)=%d\n",
         a, b, MAX(a, b));
  printf("MIN(%d,%d)=%d\n",
         b, c, MIN(b, c));
  printf("MIN3(%d,%d,%d)=%d\n",
         a, b, c, MIN3(a, b, c));
  return 0;
}
```

C konstansokról megint I

- Konstans pl.: `#define MERET 10`
- Léteznek preprocesszor utasítások, ezeket egy-egy külön sorba kell írni, a sor elejére:
  - A `#ifdef MERET` és `#endif` feltételes preprocesszor utasítások közé helyezett kódot a fordító csak akkor veszi figyelembe, ha a `MERET` definiálva van.
  - A `#ifndef MERET` és `#endif` feltételes preprocesszor utasítások közé helyezett kódot a fordító csak akkor veszi figyelembe, ha a `MERET` nincs definiálva.
  - Az `#if MERET = 5` megvizsgálja `MERET` egyenlőségét 5-tel fordítási időben, ez is `#endif` -ig tart.
  - Az `#elsif` jelenti az `else if` ágat, az `#else` pedig az `else` ágat.
  - Ha egy konstanst korábban definiáltunk, az `#undef` utasítással megszűntethetjük.

C konstansokról megint II

- Ezeknek a struktúráknak sok hasznuk van:
  - Bizonyos mértékig fordítási időben kideríthetjük, hogy milyen fordítóval dolgozunk, milyen operációs rendszeren, és ehhez igazodhatunk:
    - `#ifdef _MSC_VER`
    - `#ifdef _WIN32`
    - `#ifdef LINUX`
  - Ha egy több fájlból álló programunk van, és egy C fejléc fájlt (`.h`) több C programkód fájl(`.c`) is betölt(`#include`), illetve bizonyos függőségek miatt egyes fejlécek más fejléceket is betöltenek, akkor megakadályozhatjuk, hogy ebből gond legyen:

```
#ifndef _CHL_H_
#define _CHL_H_
// chl.c fejlec ...
static int x;
#endif /* _CHL_H_ */
```

C konstansokról megint III

- A GCC `-D` kapcsolójával megadhatunk a fordításkor egy konstanst, szintaxis:
  `gcc -Dnev=ertek`
- Példa: (`debug.c`)

```
#include <stdio.h>
int main() {
#ifdef DEBUG
   printf("debugolunk\n");
#else
    printf("nem debugolunk\n");
#endif
}
```

- Fordítás: `gcc -o debug debug.c`
  illetve `gcc -DDEBUG -o debug debug.c`

C konstansokról megint IV

Másik példa: (`trial.c`)

```
#include <stdio.h>
#define TRIAL_VERSION 1
#ifdef TRIAL_VERSION
void calculate(int a,int b){
  printf("Ez csak próbaverzió! ");
  printf("Az összes funkció eléréséhez fizess!\n");
}
#else
void calculate(int a,int b){
  printf("%d és %d számtani közepe : %f\n",
         a, b, (float)a / 2 + (float)b / 2);
}
#endif
```

C konstansokról megint V

```
int main(){
  calculate(10,20);
  return 0;
}
```

Több fájlból álló C programok I

- A programunk következő fájlokból épül fel:

  lib.h függvények deklarációja, a függvényekhez kommentek

  lib.c a `lib.h` -ban deklarált függvények implementálása

  libmain.c olyan program, amely használja a `lib` függvénykönyvtárunkat

- Elkészítés (build) a következőképpen zajlik:

```
gcc -Wall -c lib.c
gcc -Wall -c libmain.c
gcc -Wall -o lm lib.o libmain.o
```

vagy

```
gcc -Wall -o lm lib.c libmain.c
```

Több fájlból álló C programok II

lib.h:

```
#ifndef LIB_H
#define LIB_H 1
/*
 * Olyan függvény, mely az elso" paraméterében kapott
 * sztringet megfordítva
 * beleteszi a második paraméterében kapott sztringbe.
 * */
void megfordit(char *str, char *forditott);
/*
 * Olyan függvény, amely kiszámolja a paraméterében kapott
 * tömb átlagát.
 * */
float atlag(int *t, int meret);
#endif
```

Több fájlból álló C programok III

lib.c:

```
#include "lib.h"

void megfordit(char *str, char *forditott){
  int i,j;
  for(i = 0; str[i] != '\0'; i++);

  i--;

  for(j = 0; i >= 0; --i, j++)
    forditott[j] = str[i];

  forditott[j] = '\0';
}
float atlag(int *t, int meret){
```

Több fájlból álló C programok IV

```
  float atlag = 0.0;
  int i = 0;
  while(i < meret){
    atlag += *(t+i);
    i++;
  }
  atlag /= meret;
  return atlag;
}
```

Több fájlból álló C programok V

libmain.c:

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

#include "lib.h"

int main(){
  char *sz1 = "Discovery Channel";
  char *sz2 = (char*)calloc(strlen(sz1) + 1, sizeof(char));

  int tomb[] = {-2, 10, 23, -45, 67, 0, 0, 34, 99 };

  megfordit(sz1, sz2);
```

Több fájlból álló C programok VI

```
  printf("%s megfordítva : %s\n", sz1, sz2);
  free(sz2);

  printf("A tömb átlaga : %f\n", atlag(tomb, 9));

  return 0;
}
```

Feladatok

1. Készíts egy C programot, amely a parancssori paraméterként megadott számsorozatot összegzi, és a végeredményt kiírja egy fájlba! Ha nem jöttek paraméterek, jelezzen a program hibát a hiba kimeneten (`stdout`)!
2. Készíts C programot, amely másképp működik ha más néven hívjuk meg! Készíts egy szimbolikus linket és mutasd be ezt a hatást!
3. Készíts egy C programot, amely kiírja a "Home könyvtár" üzenetet, ha a home könyvtárunkból futtatjuk! Használd a `$HOME` és `$PWD` környezeti változókat!

Feladatok

További feladatok a `/pub/progalap/Gyakorlat/gyak10/` alatt.