Írta:
HECKL ISTVÁN

ADATSTRUKTÚRÁK ÉS ALGORITMUSOK PÉLDATÁR

Egyetemi tananyag

2011

LEKTORÁLTA: Dr. Fábián Csaba, Kecskeméti Főiskola Gépipari és Automatizálási Műszaki Főiskolai Kar

TÁMOGATÁS:
Készült a TÁMOP-4.1.2-08/1/A-2009-0008 számú, „Tananyagfejlesztés mérnök informatikus, programtervező informatikus és gazdaságinformatikus képzésekhez” című projekt keretében.

ISBN 978-963-279-506-5

KÉSZÜLT: a Typotex Kiadó gondozásában
FELELŐS VEZETŐ: Votisky Zsuzsa
AZ ELEKTRONIKUS KIADÁST ELŐKÉSZÍTETTE: Benkő Márta

KULCSSZAVAK:
adatstruktúra, algoritmus, adatszerkezet, adattípus, bonyolultság, programozás, feladatok, verem, sor, bináris fa, tömb

ÖSSZEFOGLALÁS:

Az adatstruktúrák és algoritmusok példatár 146 feladat segítségével segíti a hallgatók tudásának elmélyítését az adott témakörben. A példatár elején rövid áttekintés található a különböző adatszerkezetekről, az algoritmusok típusairól és bonyolultságairól. A leggyakrabban használt programozási tételek magyarázata is fellelhető itt. A feladatok típus szerint vannak csoportosítva, például veremhez kapcsolódó feladatok, bináris keresőfa feladatai, stb. A feladatok jelentős része több alfeladatból áll, amelyek sokszor önállóan is megoldhatóak. Minden feladathoz megoldás tartozik, amely a példatár végén található, C illetve C++ nyelven. A példatár az Adatstruktúrák és algoritmusok kurzuson kívül használható programozás jellegű tárgyakhoz is.

Tartalomjegyzék

1. Bevezetés

1.1. Elmélet és gyakorlat

Az „Adatstruktúrák és algoritmusok” tárgy az informatika oktatás egy fontos eleme, amely elmélyíti a programozás jellegű tárgyak ismereteit azáltal, hogy sorban megvizsgálja a gyakran használt adatstruktúrákat és a különböző algoritmusokat. Fontos, hogy az elméleti tudás mellett a hallgatók gyakorlati példák megoldása során szerezzenek tapasztalatot arról, hogy mikor, melyik algoritmus, milyen módosítással alkalmazható, annak megvalósításánál, milyen nehézségek merülhetnek fel. Ennek megfelelően a témakör elméletét tárgyaló jegyzet mellett elkészült a jelen példatár, amely számos különböző nehézségű gyakorló feladatot tartalmaz.

1.2. Kinek készül a példatár?

A példatár elsősorban B.Sc. oktatásban résztvevő műszaki informatikusok, gazdasági informatikusok, programozó matematikusok számára készül, de elkészítésekor igyekeztünk úgy eljárni, hogy a jegyzetet középiskolások is minél hasznosabban használhassák. Mint korábban említettük a példatár az Adatstruktúrák és algoritmusok elméleti jegyzetre épít leginkább. Azok használhatják leghasznosabban a jegyzetet és a példatárat, akik már rendelkeznek általános informatikai ismeretekkel és tudnak alapszinten programozni. A példák megoldása pszeudó kódban, C-ben és C++ nyelven adottak. Az Adatstruktúrák és algoritmusok számos egyéb tárgynak, például a Szoftvertechnológiának, Operációs rendszereknek, Adatbázis-kezelés elmélete, stb., az alapját fogják képezni.

1.3. Az „Adatstruktúrák és algoritmusok” jelentése

Adatstruktúrák és algoritmusok alatt sokan, sokféle témaköröket értenek. Vannak, akik idesorolják az összes összetett adatszerkezetet és a klasszikus programozás feladatok megoldását leíró programozási tételeket, mint az összegzés, keresés, eldöntés, stb. Mások e tárgy alatt speciális területeken megjelenő, nagy bonyolultságú algoritmusokat értenek, például B-fák használata, tömörítési eljárások, kódolások, hash-technikák, stb. Ekkor a feladatok gyakorlatban történő implementálása helyett, az elméleti hátterén van hangsúly. Például egy titkosítási algoritmus működésének megértése nem triviális feladat. Azzal, hogy a titkosítás megfelelő szintű-e külön tudományterület foglalkozik. Megint mások a tárgy keretében az algoritmusok bonyolultságát, azaz átlagos és legrosszabb esetben történő futási idejét és tárigényét vizsgálják.

A mi megközelítésünk az első értelmezéshez áll közel, vagyis érintjük a hagyományos programozási tételeket, de bonyolultabb témákat, például gráf algoritmusok, is tekinteni fogunk. Célunk, hogy egy alapszintű programozási ismeretekkel rendelkező hallgató, aki a C nyelv szintaktikáját (vezérlési szerkezetek, függvények, mutatók, tömbök, struktúrák) ismeri, de akinek nincsen mélyebb programozási múltja, az a példatár feladatait megoldva olyan „igazi” programozóvá váljon, aki magabiztosan használja a különféle algoritmusokat és adatstruktúrákat.

A feladatok szöveges formában adottak, így ahogy az első lépés a feladat megértése kell, hogy legyen. Tapasztalataink szerint erre külön hangsúlyt kell fektetetni az oktatás során, különben a hallgató nem tudja elkezdeni a feladatot, mert nem jön rá, hogy a megoldás során melyik algoritmust kell alkalmazni, illetve nem érti, hogyan kell egy adott adatstruktúrát az adott feladathoz igazítani. Ezt a fajta tapasztalatot nem az elmélet tanulásával, hanem gyakorlatok megoldásával lehet megszerezni. Miután valakinek két-három feladatnál kell a minimumkeresés tételét alkalmazni, ezután remélhetőleg minden köntösben felismeri a minimumkeresési feladatot és ösztönösen használja a rá tanult megoldási módszert. Ekkor már magasabb absztrakciós szinten gondolkodik az illető, mivel az összegzést, kiválasztást, stb. rutinszerűen tudja használni, ezek válnak számára egy program építőkövei, és nem az egyes változók és vezérlési szerkezetek.

1.4. Az adatstruktúra fogalma

Egy programon belül az adatokat változókban tároljuk. Az egyszerű adattípusok két nagy csoportra, egész és lebegőpontos változókra bonthatóak. Mindegyik csoportban több típus található, amelyek elsősorban a tárolható értékek értékkészletében térnek el egymástól. Például egy short int változó 2 bájton kisebb értékkészlettel rendelkezik, mint a long int, ami 4 bájtos (a legtöbb 32 bites rendszerben).

A két legfontosabb összetett adattípus a tömb és a struktúra. Az első sok ugyanolyan típusú adatot fog össze, amelyeket index alapján lehet elérni. A második több, különféle típusú változót tartalmazó változó, ahol minden tagváltozó egyedi mezőnévvel rendelkezik. Sokszor az sem teljesen egyértelmű, hogy mikor érdemes létrehozni három külön változót és mikor egy három mezőből álló struktúrát. Ha ezzel már tisztában vagyunk, akkor következő lépcsőfok, hogy az összetett adattípusokat tovább bonyolítsuk egymásba ágyazással. Például egy tömb elemei lehetnek struktúrák, amelynek az egyik mezője egy másik struktúra, amely tömböket is tartalmaz, stb.

Mutatók segítségével dinamikus adatszerkezeteket tudunk létrehozni. Legegyszerűbb esetben a mutatókat használhatjuk arra, hogy egy változót indirekt módón, mutatója segítségével érjük el. Akkor is mutatót használunk, ha egy változóról nem tudjuk előre, hogy biztosan szükség van-e rá vagy egy tömb méretét nem ismerjük fordítási időben. Ekkor a memóriafoglalás csak akkor történik meg, amikor erre explicit utasítást adunk. Ez lehetőséget ad arra, hogy például egy bekért változó értékétől függően határozzuk meg egy tömb nagyságát. A mutatók legkifinomultabb használata, amikor azok különféle viszonyokat jeleznek. Például egy láncolt listában a következő elemet határozza meg a mutató, egy bináris kereső fában pedig a szülő és a gyerek elemeket.

Adatstruktúrának nevezzük a program adatait azzal az információval együtt, hogy azok milyen egyszerű és összetett adattípusokban tároltak. Amikor adatstruktúráról beszélünk, akkor azokat az általános struktúrákat említik először, amelyet legtöbben használnak, mint a verem, sor, fa. Nem szabad elfelejteni, hogy a gyakorlatban leginkább egyedi adatstruktúrákat használunk, amelyek jóval bonyolultabbak, mint egy egyszerű láncolt lista.

Szintén fontos, hogy megértsük a különbséget az adattípus és az adatstruktúra között. Az adatstruktúra magasabb absztrakciós szinten van, mint az adattípus. Egy adatstruktúra konkrét megvalósítása a programozási nyelvtől függ, de akár egy nyelven belül is többféleképpen megvalósíthatunk egy adott adatstruktúrát. Például a bináris fa lényege, hogy minden csomópontnak legfeljebb két gyereke lehet. Ezt az adatstruktúrát megvalósíthatjuk

mutatók segítségével, úgy, hogy minden csomópontot egy struktúrával reprezentáljuk, amelynek van egy mezője, ami a bal, egy pedig, ami a jobb oldali csomópontra mutat. Ugyanez az adatstruktúra tömbbel is megvalósítható. A tömb minden eleme egy rekord és tudjuk, hogy a tömb i-dik elemének a bal oldali gyereke az 2*i-dik elemnél, a jobb oldali eleme pedig a 2*i+1-dik elemnél található. Ha a fának mondjuk, csak 3 szintje van, akkor azt is megtehetjük, hogy felveszünk 7 darab külön változót, amelyek elnevezése utal arra, hogy melyik csomópontot reprezentálják. Tehát egy darab adatstruktúrát három különféle adattípussal lehet reprezentálni. Az állítás visszafele is egy adattípussal különféle adatstruktúrákat tudunk létrehozni. Például tömb típussal reprezentálhatunk, fát, sort, vermet.

1.5. Az algoritmus fogalma

Az algoritmus informális megfogalmazása szerint egy olyan utasítássorozat, amelyet szisztematikusan követve eljutunk egy adott feladat megoldásához. Ez a definíció impliciten magába foglalja azt is, hogy csak olyan utasítás sorozat tekinthető algoritmusnak, amely véges számú lépésben megoldáshoz vezet. A végesség bebizonyítása sokszor nem triviális feladat.

A számítás illetve automata elmélet szerint a rekurzív nyelvekhez tartozó Turing gépek az algoritmusok. Az egyik Neumann elv kimondja, hogy a hardver és a szoftver egymásba átkonvertálható, vagyis egy algoritmushoz tudunk konstruálni egy vele ekvivalens Turing gépet és a Turing géphez fel tudjuk írni a neki megfelelő algoritmust.

A példatárban az algoritmus hétköznapi definícióját tekintjük, amely szerint az algoritmus egy feladat megoldásának részletes leírása vagy másképpen utasítások és vezérlési szerkezetek. Fontos hangsúlyozni, hogy egy adott feladat megoldására többfajta algoritmus létezhet. Ezek az algoritmusok különbözhetnek, tárigényben, sebességben, a megvalósítás bonyolultságában. Például egy rendezés algoritmus megvalósítható minimum kiválasztásos módszerrel quicksort és sok más módszerrel is. Mondhatjuk azt, hogy az algoritmus a probléma megoldásának a megvalósítása. Ugyanakkor az algoritmus még mindig egy absztrakt leírás, az általa tartalmazott lépések sokféle módon, például különböző programozási nyelveken, valósíthatóak meg. Ilyen megközelítésben egy program nem más, mint egy algoritmus konkrét implementálása. Az implementálás során is számos kérdésre kell választ adni. Például, ha egy halmaz elemein akarunk végigmenni, akkor az alkalmazott adatstruktúrától függ, hogy el tudjuk-e közvetlenül érni az elemeket, tudjuk-e, hogy összesen hány elem van, hogyan lehet meghatározni a következő elemet.

Az adatstruktúra és az algoritmus egészet alkot, egyiket a másik nélkül megtervezni nem lehet. Általában az adatstruktúra kialakításával kezdjük a munkát, de az adatoknak és a rajtuk végezhető műveleteknek együtt van csak értelme, ahogy ezt az objektum orientált programozás egyik elve is megfogalmazza.

1.6. Hatékonyság

Az előzőekben megállapítottuk, hogy az adatoknál és az algoritmusoknál is több szint létezik, az előbbieknél az adatstruktúra és az adattípus, az utóbbinál az algoritmus és a program szintje. Mindkét szinten felmerülnek kérdések, amelyek a hatékonyságot befolyásolják. A korábban említett minimum kiválasztós rendezés n^2-es, míg a quicksort n*log(n) hatékonyságú. Ha tudjuk, hogy osztályban a tanulók száma 30-nál sose több, akkor érde-

mes tömbben tárolni a hallgatókat, ha ellenben egyetemi évfolyamról van szó, akkor egy láncolt lista hatékonyabb lehet.

Egyáltalán miért szükséges nekünk hatékony kód, amikor a Moore törvény még mindig tartja magát, vagyis 18 havonta megduplázódnak a tranzisztorok száma a processzorokban és a sebességük töretlenül nő. Egy mai telefon tudása felülmúlja egy 20 évvel ezelőtti szuperszámítógépét. Nem kéne a hatékonyságnak, mint szempontnak háttérbe szorulnia? A válasz nem. Ennek oka, hogy a számítógépek fejlődését bőven leelőzi az elvárásaink növekedése. A megoldandó feladatok egyre nagyobbak, összetettebbek, sokrétűebbek. Minden programhoz létezik olyan nagyságú bemenet, amelyre a végrehajtási idő már percek vagy akár napok. Tehát fontos, hogy egy feladatot hatékonyan oldjunk meg, így nagyobb méretű bemenetre is használható programot kapunk.

A hatékonyságra való törekvés közben tartsuk szem előtt, hogy minden megoldás rendelkezik előnyökkel és hátrányokkal. Ha létezne egyértelműen legjobb megoldás, akkor mindenki azt használná. Például a legegyszerűbb, így leggyorsabban implementálható, legátláthatóbb kód legtöbbször a leglassabb. Első változatnak, kisméretű bemenetek esetén tökéletes, de nagyobb inputok esetén a lassúság már nem elfogadható. Szintén kompromisszumot kell kötni a sebesség és a tárigény között. Segédadatok számolásával és tárolásával a kód sebessége sokat javulhat, de erre egyre több tárterületet kell áldozni. A megoldás hatékonyságát úgy is növelhetjük, hogy az eredeti feladatnak csak egy speciális esetét vizsgáljuk, amelyre a feladat sajátosságait kihasználó kódot írhatunk. Ezután viszont az általános esetet nem tudjuk kezelni.

1.7. A példatár felépítése

A bevezetés után a feladatokat nehézség szerint próbáltuk csoportosítani. Tudjuk, hogy a nehézség fogalma viszonylagos, ez mégis valamilyen támpontot adhat a hallgatóknak. Egy feladat nehézsége több tényezőből adódhat, ezek a feladat megfogalmazása, a kód hossza, az algoritmus illetve adatstruktúra bonyolultsága, a szükséges programozási ismeretek. Egy feladat sokszor több alfeladatból áll. Az alfeladatok vagy sorban egymásra épülnek vagy egymástól függetlenek, ekkor a közös téma kapcsolja őket össze. A feladatok megoldásai a példatár végén helyezkednek el.

1.8. A fontosabb programozási tételek

A példatár használhatóságát néhány alapvető programozási tétel megadásával szeretnénk javítani. Ezek a tételek, majd minden programban megtalálhatóak ilyen vagy olyan formában. Ezek a tételek tekinthetőek a legfontosabb algoritmusoknak. Az itt látható pszeudó kódokban az első tömb index az 1.

1.8.1. Megszámlálás

A megszámlálás algoritmusa megszámolja egy adott N elemű sorozatban a T tulajdonságú elemek előfordulásainak számát. Ehhez szükségünk lesz egy változóra, amiben a számlálás eredményét tároljuk. Az algoritmus pszeudó kódja alább található. Az algoritmus minden esetben N lépést hajt végre, azaz a bonyolultsága O(N).

```
A() // Az elemeket tartalmazó tömb
N // A tömb mérete
```

```
count := 0
ciklus i := 1-től N-ig
    ha A(i) T tulajdonságú akkor
        count := count + 1
    elágazás vége
ciklus vége
kiír count
```

1.8.2. Összegzés

Az algoritmus összegzi az N elemű sorozat elemeit. A működése hasonlít a megszámlálás algoritmusra, azzal a különbséggel, hogy most az összeg változót nem egyesével, hanem a sorozat aktuális elemével növeljük meg. Az algoritmus bonyolultsága szintén O(N).

```
A() // Az elemeket tartalmazó tömb
N // A tömb mérete
sum := 0
ciklus i := 1-től N-ig
    sum := sum + A(i)
ciklus vége
kiír sum
```

1.8.3. Eldöntés

Az algoritmus eldönti, hogy egy N elemű sorozatban van-e legalább egy T tulajdonságú elem. A feladat a megszámlálás algoritmussal is megoldható, ám annak az a hátránya, hogy esetenként feleslegesen dolgozik: Ha már talált egy T tulajdonságú elemet, akkor a tömb további vizsgálata felesleges. Mivel az algoritmus elindulása előtt nem tudjuk, hogy pontosan hány elemet kell megvizsgálni, ezért számlálós helyett elöl tesztelős ciklust kell használni. A ciklus megvizsgálja, hogy az aktuális elem T tulajdonságú-e, valamint hogy nem értünk-e végig a tömbön. Amennyiben elfogytak az elemek, vagy találtunk nekünk megfelelőt, akkor a ciklus véget ér, ellenkező esetben növeljük a tömb indexet és újabb vizsgálatot hajtunk végre. Ha a ciklus után az index értéke nagyobb lesz, mint N akkor, ha nem találtunk T tulajdonságú elemet. Ha az index kisebb, vagy egyenlő, mint az N, akkor van legalább egy T tulajdonságú elem a sorozatban.

```
A() // Az elemeket tartalmazó tömb.
N // A tömb mérete
index := 1
ciklus amíg (index <= N) és (A(index) nem T tulajdonságú)
    index := index + 1
ciklus vége
ha index <= N akkor
    kiír Van legalább egy T tulajdonságú elem
egyébként
    kiír Nincs T tulajdonságú elem
elágazás vége
```

Fontos, hogy a ciklusfeltételben előbb szerepel az „index <= N”, mint az „A(index) nem T tulajdonságú” feltétel. Ellenkező esetben, ha a tömbben nincsen olyan elem, amit keresünk, akkor az algoritmus kísérletet tesz a tömb utolsó utáni elemének a megvizsgálására, ami hibát eredményez. Az algoritmus legjobb esetben 1, míg legrosszabb esetben pedig N lépést hajt végre, így a bonyolultsága O(N).

1.8.4. Szélsőérték meghatározása

Az algoritmus egy N elemű sorozatban megkeresi a minimális vagy maximális elemet. Az egyes elemeknek összehasonlíthatónak kell lenniük. Tekintsük a minimális elem keresését! Vezessük be a min_index változót, amely az algoritmus lefutása után a minimális elem indexét adja meg a sorozaton belül. Kezdetben min_index = 1, azaz ideiglenesen a sorozat első elemét tekintjük minimálisnak. Azzal a feltételezéssel élünk, hogy a tömb legalább egy elemet tartalmaz. Egy ciklussal végigmegyünk a sorozaton, és minden elemet összehasonlítunk az aktuálisan minimálisnak tekintett elemmel. Amennyiben találunk olyan elemet, amely kisebb, mint a min_index-dik elem, akkor a min_index felveszi a kisebb elem indexét. Miután végignéztük az elemeket, a min_index a legkisebb elem indexét tárolja. Maximumkeresésnél a különbség csupán annyi, hogy akkor a feltételben szereplő relációs jel irányát megfordítjuk. Az algoritmus bonyolultsága O(N).

```
A() // Az elemeket tartalmazó tömb
N // A tömb mérete
min_index := 1
ciklus i = 2-től N-ig
    ha A(i) < A(min_index)
        min_index := i
ciklus vége
kiír A legkisebb elem a min_index-edik: A(min_index)
```

1.8.5. Rendezés

Az algoritmus egy N elemű sorozat elemeit növekvő sorrendbe állítja. Számos rendező algoritmus létezik, itt a minimum kiválasztásos módszert mutatjuk be. Felhasználjuk az előző alfejezetben bemutatott szélsőérték meghatározást azzal a kiterjesztéssel, hogy megadjuk azt is, hogy a tömb mely részének a minimumát keressük. A ciklus minden iterációjában meghatározzuk a maradék, a még nem rendezett, sorozat elemei közül a legkisebbet és ezt tesszük az aktuális vagyis az i-dik helyre. Az algoritmus bonyolultsága véletlenszerű inputra átlagos esetben $O(N^2)$, de például a gyorsrendezés átlagos bonyolultsága O(n log(n)).

```
A() // Az elemeket tartalmazó tömb
N // A tömb mérete
ciklus i = 1-től N-1-ig
    mi := min_index_keresés(A, i, n)
    cserél(A(i), A(mi))
ciklus vége
kiír A
```

2. Általános feladatok

2.1. Vizsgaeredmény

2.1.1. Írjon programot, amely bekéri egy vizsga eredményének százalékos értékét és kiírja, hogy sikeres volt-e a vizsga! A siker feltétele az 50%-nál jobb teljesítmény. Ötlet: ügyeljen arra, hogy az 50% még bukást jelent!

2.1.2. Kérjen be pontszámot, amelynek -25 és 50 közé kell esnie! A vizsga sikerességét csak akkor vizsgálja, ha a pontszám a helyes intervallumban van! A siker feltétele a 25 pontnál jobb eredmény.

2.2. Sakktáblarajzolás

Írjon programot, amely egy üres sakktáblát rajzol a karakteres képernyőre. A sakktábla minden mezője két egymás melletti szóközből álljon! A mezők elválasztására használja a '-' és '|' karaktereket. Készítsen fejlécet a sorok és az oszlopok számára is, a soroknál számokat, az oszlopoknál betűket használjon! A programot később egy sakkprogram egyik komponense lesz.

2.2.1. A sakktábla elkészítéséhez csak printf függvényt használjon a programban! Ötlet: minden sorhoz külön printf-t használjon!

2.2.2. Az azonos sorok kiírásához használjon ciklust!

2.2.3. Egy adott sor kirajzolásához is használjon ciklust! Keresse meg egy sor ismétlődő mintáját és az kerüljön a ciklus magjába! Figyeljen különösen a fejlécek helyes megjelenítésére.

2.3. Terület és kerületszámítás

2.3.1. Írjon programot, amely bekéri egy téglalap két oldalának a hosszát, és hogy területet vagy kerületet akar-e a felhasználó számolni! A választástól függően számolja ki a területet vagy a kerületet! Az oldal hosszak egész értékek, a választás karakter típusú. Ötlet: ügyeljen arra, hogy negatív nagyságú oldal nincsen!

2.3.2. Módosítsa az előző programot úgy, hogy az oldalak hossza tört érték is lehessen! A program elején kérje be, hogy négyzet vagy téglalapot vizsgál-e és ennek függvényében kérjen be egy vagy két oldal hosszat!

2.3.3. Bővítse ki a programot úgy, hogy az oldalak hossza csak pozitív érték lehet, síkidomválasztáskor a kis és a nagy betűket is fogadjuk el, ha rossz választ adtunk, akkor az jelezzük, vegyük be a választható idomok közé a szabályos háromszöget!

2.4. Számláló ciklus

2.4.1. Írjon programot, amely kiírja az egész számokat 0-tól 9-ig!

2.4.2. Módosítsa az előző programot úgy, hogy 40-től 60-ig fordított sorrendben, a páros számokat írja ki! Az intervallum mindkét határát írja ki!

2.5. Sebességmérés

2.5.1. Egymásba ágyazott ciklusok segítségével határozza meg, hogy hány üres iteráció tesz ki egy másodpercet, ha a ciklusváltozó egész, illetve ha lebegőpontos! Minden ciklus 1000 lépést tegyen meg! Hány egymásba ágyazott ciklus kell, hogy a végrehajtási idő 5-10 másodperc legyen?

2.5.2. Határozza meg az összeadás, szorzás, osztás, hatványozás időigényét úgy, hogy az előző program legbelsejébe beírja a megfelelő utasítást!

2.6. Adatbekérés

2.6.1. Írjon programot, amellyel bekéri egy ember adati közül a következőket: azonosító szám - int, életkor - unsigned int, neme - char, súlya - float, bankszámla egyenlege - double, neve - char[200]! A bekért adatokat írja ki újból a képernyőre! A megadott és a kiírt érték mikor különbözhet? Mutasson rá példát!

2.6.2. Kérje be majd írassa ki egy autó tulajdonságait! A tulajdonságok a következőek: márka - char[200], végsebesség - float, regisztrációs szám - hexa, tömeg - double, típus név - char[300], utas szám - int, műholdas navigáció - char.

2.7. Blokkon belüli változók

2.7.1. Hozzon létre egy olyan main függvényt, amelyben 3 különböző típusú, 'a' nevű változó szerepel! Az egyes változók típusai: const float, char [200], unsigned long int. Ugyanolyan nevű változókat egy függvényben belül úgy tud létrehozni, hogy a változó definiálások különböző blokkokban vannak. Egy negyedik változóba kérje be, melyik blokkba menjen be a program, írassa ki az ott lévő a változó értékét!

2.8. Formázott kimenet

2.8.1. printf segítségével írja ki a következő sorokat, úgy hogy változókban tárolja az egyes értékeket! A szóközt „_”-al jelöljük. A kiírandó szöveg:

```
_+12.45
_-234.1
+57967.2
+134567
```

2.8.2. Az előző feladathoz hasonlóan írassuk ki a következő sorokat:

```
_ _+235
_-1.291E4
_00026
0XAB2
1.64e-003
```

2.9. Hagyományos vagy normál forma

2.9.1. Írjon programot, amely öt darab tömbben tárolt float típusú számról kérdezi meg, hogy azt a printf %g-el történő kiírásakor hagyományos (f) vagy normál (e) alakban jeleníti meg! A számot jelenítsük meg először %f-el! Írjuk ki a felhasználónak, hogy tippeljen, jelenítsük meg a választ %g-el, kérdezzük meg, hogy helyes volt-e a tipp! Számoljuk a helyes válaszokat!

2.9.2. Alakítsuk át a programot úgy, hogy ne a tipp helyességét kérjük be, hanem magát a tippet! Használjunk sprintf parancsot és használjuk ki azt, hogy normál forma mindig tartalmaz egy 'e' betűt. A program elején kérjük be a tömb nagyságát, a program végén, %-ban adjuk meg a helyes megoldások számát!

2.9.3. Alakítsuk át a programot úgy, hogy minden iterációban kérje be a %g-ben használt pontosságot is! A válasz ne csak az legyen, hogy hagyományos vagy normál formában történik a kiírás, hanem pontosan meg kell adni számot a megfelelő formában!

2.10. Szorzótábla

2.10.1. Készítsünk 5*5-ös egyedi szorzótáblát, amelyet a karakteres képernyőn jelenítünk meg! Az első sor és oszlop értékeit, amelyek mutatják, hogy mit mivel kell összeszorozni, vonalakkal különítsük el a szorzatoktól! Az első sor és az első oszlop értékeit a billentyűzetről kérjük be!

2.10.2. A szorzótáblához hasonló módon valósítsuk meg az egyedi összeadó táblát! Az első sor és az első oszlop értékeit szöveg fájlból olvassuk be!

2.11. Másodfokú egyenlet

2.11.1. Írjon programot a másodfokú egyenlet megoldására! Először kérje be az együtthatókat, azután írja ki, hogy hány megoldás van és adja meg a megoldásokat!

2.11.2. Határozza meg a komplex gyököket is! A komplex gyököket is figyelembe véve a másodfokú egyenletnek mindig két megoldása van, de azok egybeeshetnek.

2.12. Operátorok

2.12.1. Írjon programot a relációs operátorok működésének a szemléltetésére! Kérjen be két egész változót és írja ki, hogy az egyes relációs operátorokat alkalmazva rájuk milyen eredményt kapunk. Legyen lehetőség az előző lépések ismétlésére!

2.12.2. Írjon programot a logikai és bitenkénti operátorok használatának a szemléltetésére!

2.13. Maszkolás

2.13.1. Kérjen be egy előjel nélküli karakter típusú változót egészként és írja ki a kettes számrendszerbeli alakját!

2.13.2. Kérjen be egy számot és az előző programmal írja ki a kettes számrendszerbeli alakját! Kérjen be egy bit pozíciót és állítsa azt a bitet egyesre! Írja ki az új értéket tízes és kettes számrendszerben is!

2.13.3. Folytassa az előző programot úgy, hogy bekér még két bit pozíciót! Az elsőnél törölje a bitet a másodiknál negálja. Minden részeredményt írjon ki a képernyőre!

2.14. Hatványsor

2.14.1. Írjunk programot, amely hatványsor és könyvtári függvény segítségével is kiszámolja e^x értékét a következő képlettel: $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$ Megfelelően pontos értéket kapunk, ha a sort a negyedik elemig határozzuk meg. Írjuk ki a hatványsor és a valós érték közötti különbséget!

2.14.2. Kérjünk be egy pontosságot és határozzuk meg, hogy a sor hány tagját kell figyelembe venni, az adott pontosság eléréséhez!

2.14.3. Határozzuk meg a sin(x)-t hatványsor segítségével! $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$

2.15. Típuskonverzió

2.15.1. Írjon programot, amely bekér egy double értéket kiírja az eredeti értéket a float, int, short int, char típussá konvertált változatok nagyságát és értékeit! A változók méretei jobbra legyenek rendezve és az értékek kiírása azonos oszlopban kezdődjön!

2.15.2. Határozza meg az eredeti és a konvertált értékek közötti különbségeket!

2.16. DeMorgan

2.16.1. Írjon programot, amelyben egész változókat használ logikaiként! A változók a következőek: a - van pénzem, b - van kedvem, c - nincs időm, d - nagyon érdekel. Kérje be a változók értékeit és a következők alapján döntse el, hogy megy-e nyaralni: ha van pénze, kedve és ideje, akkor megy, ha az előzőek közül csak egyik igaz, de nagyon érdekli akkor is megy. Írja át a feltételeket DeMorgan azonosságok segítségével!

2.16.2. Kérje be a következő változókat: a - esik az eső, b - jönnek mások is, c - van szabadnapom. Ezen változók alapján határozzuk meg, hogy mikor megy a felhasználó túrázni. A feltételek a következőek: akkor megyek túrázni ha, nem esik az eső és van szabadnapom; akkor is túrázom, ha mások nem jönnek, de az eső esik. Írja át a feltételeket De Morgan azonosságok segítségével!

2.17. Halmazállapot

2.17.1. Írjon programot, amely a víz hőmérséklete alapján megállapítja annak halmazállapotát!

2.17.2. Alakítsa át a programot, hogy ne Celsius fokot, hanem Farenheit-t kér be. Külön függvény végezze el a konvertálást!

2.18. Római számok

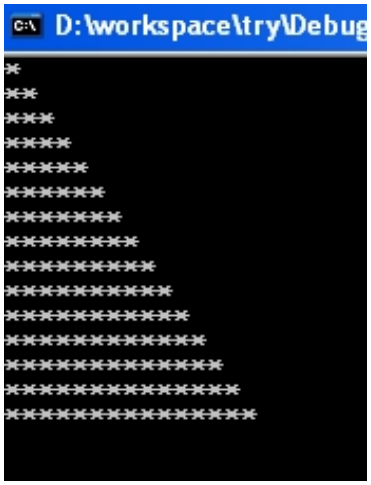
2.18.1. Írjon programot, amely megadja, hogy mik a római számjegyek (I, V, L, C, D, M) arab megfelelői! Kérjen be egy római számjegyet, konvertálja nagybetűre, ha szükséges és adja meg a szám értéket!

2.18.2. Ne csak egy darab számjegyet, hanem egy több karakterből álló számot alakítson át!

2.18.3. Valósítsa meg a visszafele történő konverziót!

2.19. Háromszög

2.19.1. Rajzoljon ki a karakteres képernyőre egy derékszögű, egyenlő szárú háromszöget csillagokból dupla for ciklus segítségével!



2.1. ábra: Lehetséges képernyőkép

2.19.2. Fordítsa el a háromszöget úgy, hogy a másik befogó kerüljön alulra!

2.20. Négyzet animáció

2.20.1. Készítsen a karakteres képernyőn videót egy leeső négyzetről! A négyzet 3*3 csillagból álljon! Miután megjelenített egy képet, várjon rövid ideig, törölje a képernyőt és jelenítse meg a következő képet egy sorral lejjebb! Feltesszük, hogy a karakteres képernyő 80*25-ös.

2.20.2. Módosítsa úgy a programot, hogy amint a négyzet eltűnik alul, felül jelenjen meg újból!

2.20.3. Módosítsa úgy a programot, hogy négyzet helyett háromszög jelenjen meg, ami alulról pattanjon vissza!

2.20.4. Négyzet helyett téglalap mozogjon vízszintesen a szélekről visszapattanva!

2.21. Legnagyobb közös osztó

2.21.1. Írjon programot, amely kiszámolja két szám legnagyobb közös osztóját a következő algoritmus segítségével!

```
ciklus amíg a két szám nem egyenlő
    a nagyobb szám értékét csökkentsük a kisebb számmal
```

2.21.2. Módosítsa úgy a programot, hogy az három szám legnagyobb közös osztóját számolja ki!

2.22. main paraméterek

2.22.1. Írjon programot, amely kiírja a program paramétereit fordított sorrendben!

2.22.2. Írja át úgy a programot, hogy elől tesztelős ciklust használ és a program nevét nem írja ki!

2.23. continue, break

2.23.1. Írjon programot, amely osztást végez el egy hátul tesztelős ciklusban! Használjon végtelen ciklust, amelyet majd break paranccsal fog megszakítani! Kérje be a két változót! Ha az osztandó abszolút értéke nagyobb, mint 100, akkor hiba üzenet után hajtsa végre a következő iterációt! Ha az osztó értéke 0, akkor szakítsa meg a ciklust! Szintén szakítsa meg a ciklust, ha már három osztást elvégzett!

2.24. Deriválás

2.24.1. Írjon függvényt és hozzá tartozó programot, amely a $3x^3-2x^2+6x-1$ függvénynek megadja az 1., 2. vagy 3. deriváltjának az értékét egy adott pontban! A függvény paraméterei: hányadik deriváltról van szó, milyen pontban tekintjük a deriváltat. A függvény visszatérési értéke: a derivált értéke.

2.24.2. A sin(x) függvénynek határozza meg az 1., 2. vagy 3. deriváltjának az értékét egy adott pontban! Használja fel, hogy sin'(x) = cos(x), cos'(x) = -sin(x).

2.24.3. Tetszőleges negyed-fokú polinomnak határozza meg a deriváltját adott pontban! Legyen a függvénynek egy harmadik paramétere, ahol jelezzük, ha hiba történt, vagyis ha az első paraméter értéke helytelen, vagyis négynél nagyobb!

2.25. Lépésenkénti összegzés

2.25.1. Írjon programot, amely a konzolról kér be egész számokat egy ciklus segítségével és egy függvény segítségével számolja ki azok összegét! Ezt a függvényt minden szám bekérése után hívja, ami ezután visszaadja az eddigi számok összegét. Használjon statikus változót a részösszeg tárolására!

2.25.2. Írjon olyan függvényt az előző programhoz, amely lenullázza az aktuális összeget!

2.25.3. Módosítsa úgy az előző programot, hogy nem használ statikus változót, hanem paraméterként adja át az eddigi részösszeget!

2.26. Alias változók

2.26.1. Írjon programot, amelyben egy kocka felületét és térfogatát számolja ki! Úgy írja fel a képletet, hogy két ugyanolyan nevű változó ne szerepeljen benne, hanem használjon alias változókat mutatók segítségével.

2.26.2. Demonstrálja, hogy a * és a & ellentétes hatású műveletek!

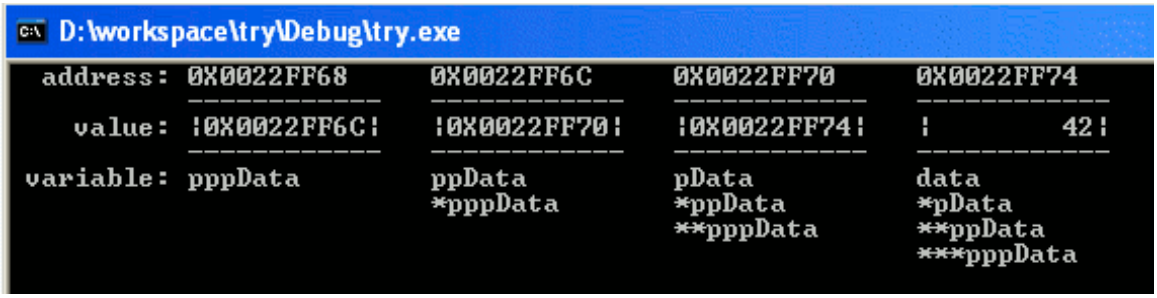
2.26.3. Írjon az előzőhöz hasonló programot gömbre vonatkozóan!

2.27. Két kulcsos hozzáférés

2.27.1. Írjon programot, amely egy titkos adatnak (változónak) a címét egy mutatóba és egy egészbe kódolja úgy, hogy a mutató és az egész összege a változó címét eredményezi! A program adja meg a két adatot két embernek egyenként. Később kérje be a mutatót és az egészet és ha őket összeadva visszakapjuk a változó címét, akkor írja ki a változó értékét.

2.28. Többszörös indirekció

2.28.1. Írjon programot, amely a konzolos képernyőn szemléltet egy int*** mutatót és az általa mutatott értékeket! Írja ki a jelenlévő változók címét, értékét és hogy milyen alternatív módon lehet hivatkozni a változókra!



2.2. ábra: Lehetséges képernyőkép

2.28.2. Írjon programot, amely a konzolos képernyőn szemléltet egy 2*3 dinamikus double tömböt!

2.29. Típusválasztás

2.29.1. Írjon programot, amelyben bekéri, hogy rövid egész vagy hosszú lebegőpontos típussal akar dolgozni. Hozzon létre két dinamikus változót a megfelelő típusból! Kérje be a változók értékeit és az első változó értékét növelje meg a másodikéval!

2.30. Struktúra kezelés

2.30.1. Készítsen struktúrát, amely minden egyszerű adattípusból tartalmaz egyet valamint egy egészre mutató mutatót! Kérje be a struktúra adattagjait a konzolról, majd írja ki azokat!

2.30.2. Módosítsa az előző programot úgy, hogy létrehoz új típusnevet a struktúrának és ezt használja a továbbiakban! Külön függvényben valósítsa meg a struktúra bekérő és kiíró részt! Hozzon létre egy dinamikus struktúrát, amit tegyen egyenlővé a bekért struktúrával! Módosítsa a dinamikus struktúrán keresztül az egész mutató által mutatott értéket!

2.31. Esküvő

2.31.1. Készítsen programot, amely tartalmazza a következő adatszerkezeteket: ember típus: név, életkor, azonosító; gyűrű típus: karát, érték, kövek száma; esküvő típus: férj, feleség, eljegyzési gyűrű, esküvői gyűrű, vendégek (dinamikus tömb), vendégek száma. Inicializáljon egy esküvő típusú változót, írja ki a változó értékét szépen tördelve úgy, hogy minden struktúra kiíráshoz külön függvényt hoz létre!

2.32. Ventilátor

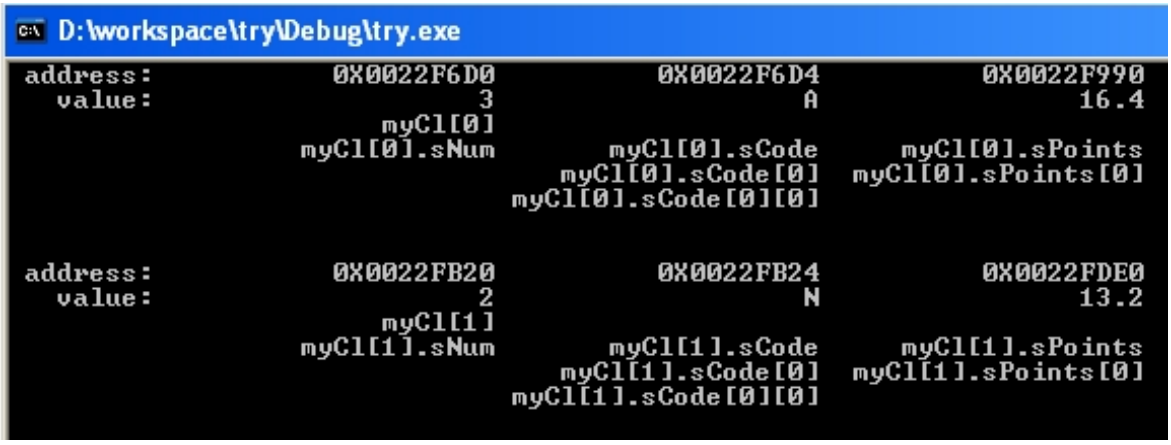
2.32.1. Írjon programot, amelyben létrehoz egy ventilátor típust: gyártó (statikus tömb), termékszám, leírás (dinamikus tömb), ár! Írjon függvényt, amely bekéri az

adattagok értékét és a rekordot visszatérési értékként adja vissza! Szintén írjon függvényt a struktúra kiírásához!

2.32.2. Módosítsa úgy a programot, hogy a ventilátor változó értékét tegye egyenlővé egy másik változóval, módosítsa az eredeti változóban a gyártót és a leírást! Írassa ki újból mindkét struktúrát! Milyen furcsaságot tapasztal? Hogy lehet azt korrigálni?

2.33. Többszörösen összetett típus

2.33.1. Írjon programot, amelyben szemlélteti a karakteres képernyőn a struktúrák elhelyezkedését a memóriában! Hozzon létre egy osztály típusú struktúrát: hallgatók száma, hallgatók Neptun kódjai (statikus sztring tömb), hallgatók pontszámai (statikus lebegőpontos tömb)! Írjon függvényt, amely megjelenti a paraméterként átadott struktúra adattagjainak a címeit és értékeit (a tömb típusú mezőknek csak az első elemét kell megjeleníteni)! Írja ki azt is, hogy milyen más módon tud az adott címre hivatkozni! Ezt az információt is paraméterként adja át a kiíró függvénynek! Hozzon létre kételemű tömböt osztály típusú struktúrákból! Inicializálja a tömböt és hívja meg mindkét elemre a kiíró függvényt!



2.3. ábra: Lehetséges képernyőkép

2.34. Bitmezők

2.34.1. Írjon programot egyén adatainak a tárolásához! Az ember típus mezői a név (statikus tömb), nem, nagykorú-e, csillagjegye, vallása (5 nagy vallás vagy egyéb), élő személy-e, vércsoport, RH csoport! Használjon megfelelő hosszú bitmezőket! Írjon bekérő és kiíró függvényt, a kiíró függvénynél ne számokat, hanem azok jelentéseit írja ki! Határozza meg a struktúra méretét!

2.34.2. Módosítsa úgy a struktúrát, hogy bitmezők helyett egész változókat használ! Mennyivel nő a struktúra mérete?

2.35. Szóbeli vizsga

2.35.1. Írjon programot, amelyben bekéri egy szóbeli vizsga minősítését! A minősítés lehet: szörnyű, rossz, gyenge, jó, kiváló. Használjon felsorolás típust! Írjon függvényt a minősítés szöveges kiírására! Kérje be az ismétlő vizsga eredményét, határozza meg, hogy javított-e az illető!

2.35.2. Oldja meg az eredeti feladatot #define-ok segítségével! Vezesse be a közepes minősítést! Miért jobb felsorolás típust használni define helyett?

2.36. Csak olvasható fájl

2.36.1. Írjon programot, amely beállítja egy fájl „csak olvasható” tulajdonságát! Kérje be egy szöveges fájl nevét! Állítsa be a fájl csak olvasható tulajdonságát, majd próbáljon meg a fájl végéhez fűzni egy szöveget C-ből hívott DOS parancs segítségével (DOS parancs: echo text >>fileName)! Futtassa a DOS parancsot C-ből: system(char*) segítségével! Törölje a csak olvasható tulajdonságot és próbáljon ismét a fájl végéhez fűzni egy szöveget!

2.36.2. Írja meg a tuti törlés programot! A program először törli a csak olvasható tulajdonságot és utána DOS parancs segítségével törli a fájlt.

2.37. Átnevezés

2.37.1. Feladat: Írjon programot fájlok átnevezésére! A régi és az új fájlnév a program paraméterében legyen adott! Ha valamelyik paraméter hiányzik, akkor írja ki a helyes használatot! Ha hiba történik átnevezés közben, akkor írja ki a megfelelő hibaüzenetet a hibakód alapján!

2.38. Szövegfájl

2.38.1. Írjon programot, amely bekéri a következő információkat: minimális érték, maximális érték fájl név, adatok száma! Generáljon adott számú véletlen számot a megfelelő határok között és mentse el azokat egy szövegfájlba külön sorokba!

2.38.2. Módosítsa úgy a programot, hogy az első sorba írja ki az adatok számát! Szóközökkel válassza el az egyes értékeket és minden számot fix hosszan írjon ki! 100 számonként szúrjon be egy üres sort!

2.39. Hamupipőke

2.39.1. Írjon programot, amely egy már meglévő, véletlen számokat tartalmazó fájlban lévő számokat szétválogatja párosság alapján! A páros és páratlan számokat tárolja külön fájlokban! Írjon függvényt, amely kiszámolja az egy fájlban lévő számok átlagát! Alkalmazza a függvényt a két fájlra és határozza meg a két fájlban lévő számok mennyiségének az arányát!

2.40. XML

2.40.1. Írjon programot, amely XML-ben ment el rekordokat! A kocsi típus azonosítót és árat, a személy típus nevet, életkort, a kocsik számát, és kocsi tömböt tartalmaz. Hozzon létre és inicializáljon egy 3 hosszú személy tömböt és mentse el azokat XML fájlba! Az XML fájl legyen szépen tördelve! Írjon halfTag és fullTag függvényeket! Az előbbi csak „<tagName>” vagy „</tagName>” -t ír ki, az útóbbi „<tagName> Text </tagName>”-t ír ki.

Példa:

```
<Person>
     <Name>Jani</Name>
     <Age>22</Age>
     <Count>3</ Count>
     <Car>
        <ID>0</ID>
        <price>50.000000</price>
     </Car>
     <Car>
     …
```

2.40.2. Módosítsa úgy az XML fájlt, hogy Car tag-k egy Cars tag-on belül legyenek! Ahol lehet, az adatok a tagok tulajdonságaiban szerepeljenek! Például: <name value="Jani"/>

2.40.3. Készítsen beolvasó programot! Jelezze ki az XML fájl szintaktikai hibáit! Például hibás tag, nyitó tag párja hiányzik vagy fordítva, két tag pár keresztben van.

2.41. Stopper

2.41.1. Készítsünk stopperórát, amely a program indításától eltelt időt írja ki folyamatosan! A program 11 másodperc után álljon meg!

2.41.2. Módosítsa a programot úgy, hogy csak minden egész másodpercben írja ki az eltelt időt!

2.42. Időpont formázása

2.42.1. Írjon programot, amely 10 másodpercen keresztül folytonosan kiírja az aktuális időt ezredmásodperc pontossággal! Használja az _ftime és ctime függvényeket!

2.43. Tippelés

2.43.1. Írjon időtippelő programot! A program állítson elő egy véletlen számot 1 és 5 között és ezt írja ki a képernyőre! Ennyi időt kell majd várni a felhasználónak.

Ezután kérjen be két sztringet úgy, hogy méri a bekérések között eltelt időt! Írja ki, hogy ténylegesen mennyi idő telt el a két bekérés között! Ha a bemenet a „quit” sztring, akkor lépjen ki a programból!

2.44. Milyen nap?

2.44.1. Írjon programot, amely véletlenszerűen meghatároz egy évet, hónapot, napot és utána tippelni kell, hogy az a hét illetve év napjai közül a hányadik! Adja meg hogy helyesek voltak-e a tippek!

2.45. Hérón képlet

2.45.1. Írjon függvényt, amely a Hérón képlet segítségével kiszámolja a háromszög területét! Dobjon float típusú kivételt, ha valamelyik oldal változó negatív és dobjon const char* típusú kivételt, ha nem teljesül a háromszög egyenlőtlenség! A kivételeket kezelje a main függvényben!

2.46. Háromszög

2.46.1. Készítsen derékszögű háromszög osztályt, amelyben az adattagok az oldalak! Írja meg a set és get függvényeket az adattagokra! Írja meg az area, perimeter, check (ellenőrzi, hogy tényleg derékszögű-e a háromszög) és display (kiírja az oldalak hosszát) függvényeket!

2.46.2. Készítsen hasonló osztályt egyenlő szárú háromszögre!

2.46.3. Készítsen hasonló osztályt szabályos háromszögre!

2.47. Statikus adattag

2.47.1. Készítsen osztályt, amelynek segítségével fájlba tud írni sorokat! A fájlnevet tudjuk beállítani objektumonként és statikus adattag számolja a program által eddig kiírt sorok számát! Lehessen ezt az adattagot lenullázni és lekérdezni!

2.47.2. Módosítsa az előző programot úgy, hogy számolja a szavak és karakterek számát is!

2.48. Komplex számok

2.48.1. Írjon komplex szám osztályt! Használjon privát adattagokat! Írja meg az add, sub, mul, div metódusokat! Az említett metódusok írják ki a műveletek eredményét a képernyőre!

2.48.2. Módosítsa úgy a programot, hogy a metódusok ne végezzenek kiírást, hanem adják vissza az eredményt objektumként! Valósítsa meg kiíró operátort a komplex osztályra!

2.48.3. Valósítsa meg az aritmetikai függvényeket operátorok segítségével!

2.49. Telefonszámla

2.49.1. Írjon programot egy telefontársaság számára, ami az egyes előfizetők telefonszámláit szeretné kiszámolni! Az előfizetők adatai az „in.txt” nevű fájlban tárolja! A program készítsen felsorolást minden előfizető számára, amelynek tartalmaznia kell a teljes beszélgetési időt, és a fizetendő összeget! Ezután a program számolja ki, melyik előfizető telefonált összesen a legtöbb ideig, ez milyen hosszú idő, kinek kell a legtöbbet fizetni, és mennyit! Használjon dinamikus tömböket! Feltételezzük, hogy a bemeneti fájl az előírásoknak megfelelő.

Az input formátuma:

Első sor: az előfizetők száma
Második sor: két tízes számrendszerbeli szám, a csúcsidőben való telefonálás tarifája, és a csúcsidőn kívüli telefonálás tarifája.
A többi sor az előfizetők adatait tárolja: az előfizető neve (keresztnév, vezetéknév), majd két tízes számrendszerbeli szám: mennyi időt telefonált az illető csúcsidőben, illetve csúcsidőn kívül. Mind a keresztnév, mind a vezetéknév maximum 10 karakterből áll.

Példa bemenet:

```
2              // két előfizető van
3 1            // a hívásdíj 3 Ft/perc csúcsidőben, és 1 Ft/perc
csúcsidőn kívül
Bela Kovacs 731 123  // Kovács Béla 731 percet telefonált csúcsidőben, és
123 percet csúcsidőn kívül

Mihaly Toth 300 35
```

Kimenet:

```
Bela Kovacs:              2316 Ft   854 minute
Mihaly Toth:              935 Ft   335 minute

Highest fee: 2316 Ft
        Bela Kovacs
Longest speaking: 854 Ft
        Bela Kovacs
```

2.50. Halmazok metszete

2.50.1. Írjon programot, amely két halmaz metszetét generálja! Mindkét halmaz egész számokat tartalmaz. A program olvassa be az input fájlt! Kérjen be egy számot a billentyűzetről, és döntse el, hogy a szám eleme-e az első halmaznak, a második halmaznak, illetve ezek metszetének! Írassa ki a képernyőre az előbbi három halmazt! Végül írassa ki, hogy az első halmaz részhalmaza-e a másodiknak, illetve fordítva! Használjon dinamikus tömböket!

Az input fájl formátuma:

Első sor: az első halmaz mérete
Második sor: az első halmaz elemei
Harmadik sor: a második halmaz mérete
Negyedik sor: a második halmaz elemei
A számok egy-egy szóköz karakterrel vannak elválasztva.

Példa bemenet:

```
3
80 54 41
5
41 21 54 76 80
```

Kimenet:

```
Type a number: 76
76 is not element of A
76 is element of B
76 is not element of intersection of A and B
A = {80, 54, 41}
B = {41, 21, 54, 76, 80}
Intersection of A and B = {41, 54, 80}
B is not subset of A
A is subset of B
```

2.51. Halmazok uniója

2.51.1. Írjon programot, amely két halmaz unióját generálja! Mindkét halmaz egész számokat tartalmaz! A program olvassa be az input fájlt. Kérjen be egy számot a billentyűzetről, és döntse el, hogy a szám eleme-e az első halmaznak, a második halmaznak, illetve ezek uniójának! Írassa ki a képernyőre az előbbi három halmazt! Végül írassa ki, hogy az első halmaz részhalmaza-e a másodiknak, illetve fordítva! Használjon dinamikus tömböket!

Az input fájl formátuma:

Első sor: az első halmaz mérete
Második sor: az első halmaz elemei
Harmadik sor: a második halmaz mérete
Negyedik sor: a második halmaz elemei
A számok egy-egy space karakterrel vannak elválasztva.

Példa bemenet:

```
3
80 54 41
5
41 21 54 76 80
```

Kimenet:

```
Type a number: 76
76 is not element of A
76 is element of B
76 is element of union of A and B
A = {80, 54, 41}
B = {41, 21, 54, 76, 80}
Union of A and B = {41, 21, 54, 76, 80}
B is not subset of A
A is subset of B
```

2.52. Halmazok különbsége

2.52.1. Írjon programot, amely két halmaz különbségét generálja! Mindkét halmaz egész számokat tartalmaz. A program olvassa be az input fájlt! Kérjen be egy

számot a billentyűzetről, és döntse el, hogy a szám eleme-e az első halmaznak, a második halmaznak, az A\B vagy a B\A halmaznak! Írassa ki a képernyőre az előbbi négy halmazt! Végül írassa ki, hogy az első halmaz részhalmaza-e a másodiknak, illetve fordítva! Használjon dinamikus tömböket!

Az input fájl formátuma:

Első sor: az első halmaz mérete
Második sor: az első halmaz elemei
Harmadik sor: a második halmaz mérete
Negyedik sor: a második halmaz elemei
A számok egy-egy szóköz karakterrel vannak elválasztva.

Példa bemenet:

```
3
80 54 41
5
41 21 54 76 80
```

Kimenet:

```
Type a number: 76
76 is not element of A
76 is element of B
76 is not element of A \ B
76 is element of B \ A
A = {80, 54, 41}
B = {41, 21, 54, 76, 80}
Set difference A\B = {}
Set difference B\A = {21, 76}
B is not subset of A
A is subset of B
```

2.53. Binomiális tétel

2.53.1. Írjon programot, amely beolvas egy *n* értéket a billentyűzetről, majd a binomiális tétel jobb oldalának kifejtett alakját kiírja a képernyőre! *n* ne legyen nagyobb 10-nél! A binomiális tétel szerint:

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$$

ahol *n* pozitív egész szám, és $(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}$

Példa bemenet:

n=3

Kimenet:

```
( a + b )^3 = a^3 + 3ba^2 + 3b^2a + b^3
```

2.54. Hazárdjáték

2.54.1. Írjon hazárdjátékot! A szabályok a következők: A játék kezdetekor a játékosnak 1000 Ft-ja van. A program generál egy véletlen számot, a játékosnak pedig meg kell adnia egy tippet, és egy tétet. Ha eltalálja a számot, megkapja a tét összegét, ha nem, elveszíti. Ezután a program megkérdezi: „Try again? (y/n)”, a játékosnak pedig egy „y” vagy egy „n” karakterrel kell válaszolnia. Ha újból próbálkozik, akkor a program új számot generál. A játék addig folytatódik, amíg a játékosnak van pénze, vagy amíg nem szeretné befejezni a játékot. 3 különböző szint van: a „könnyű”, amelyben a program 1 és 5 közötti számot generál, a „közepes”, amelyben 1 és 10 közötti számot generál, és a „nehéz”, amelyben 1 és 15 közötti számot generál. A szintet a játék megkezdése előtt választja ki a játékos, egy egyszerű menü segítségével. A játék végén a program írja ki a képernyőre, mennyit nyert vagy veszített a játékos.

Példa játék:

```
Choose the difficulty!

        Easy:..........1
        Medium:.........2
        Hard:..........3

2

You have 1000 HUF now.
Take your stake: 10
What's your tip (1..10)? 5
You've hit the number!
You have 1010 HUF!
Try again? (y/n)
y

Take your stake: 1000
What's your tip (1..10)? 1
The number was 7
You have 10 HUF!
Try again? (y/n)
n

Congratulations!
You get 10 HUF!
```

2.55. Dátumellenőrzés

2.55.1. Írjon programot, amely beolvas egy dátumot egy fájlból, szintaktikai és szemantikai ellenőrzést végez rajta és kiírja az esetleges hibákat! Szintaktikai ellenőrzés: A dátum 3 számot tartalmaz, mindegyik szám után egy pont szerepel. A számok csak számjegyet tartalmazhatnak! Például a „2007.” megfelelő, de a következők nem: „2A07.”, „2007”. Szemantikai ellenőrzés: Akkor hajtódik végre, ha a szintaktikai ellenőrzés nem jelzett hibát. A hónap értéke 1 és 12 közt, a napé 1 és 31 közt változhat. Ha az értékek valamelyike hibás, ezt a program írja ki a képernyőre.

Az input fájl formátuma:

Év. Hónap. Nap.

Példa bemenet:

```
2007. 12. 31.
```

Kimenet:

```
2007. 12. 31.
*****************************
I found 0 errors
```

Példa bemenet:

```
2007 1u2. 31
```

Kimenet:

```
2007 1u2. 31
Error: A "." have to follow the number, not "7"!
Error: the Month cannot contain u character!
Error: A "." have to follow the number, not "1"!
*****************************
I found 3 errors
```

Példa bemenet:

```
2007. 13. 0.
```

Kimenet:

```
2007. 13. 0.
Error: There are only 12 months!
Error: There are at most 31, at least 1 days in a month!
*****************************
I found 1 errors
```

2.55.2. Írja át az előző programot úgy, hogy csak szemantikai ellenőrzés legyen, de abból szigorúbb! Az év 1000 és 3000 közti érték lehet. A hónap 1 és 12 közti érték lehet. A nap értéke 1 és 31 között lehet januárban, márciusban, májusban, júliusban, augusztusban, októberben és decemberben. A nap értéke 1 és 30 közti értéket vehet fel áprilisban, júniusban, szeptemberben és novemberben. Ha az év szökőév, februárban a nap értéke 1 és 29 közt lehet, ha nem, akkor 1 és 28 között. Szökőév minden 4-gyel osztható év, de a 100-zal oszthatóak nem. A 400-zal osztható évek mindig szökőévek. Például 1992, 1996, 2000, 1600 szökőév, de 1800, 1900 nem az. A program elsőként az évet és a hónapot ellenőrizze, majd ha ezek megfelelőek, akkor a napot is! Írja ki a képernyőre a felfedezett hibákat!

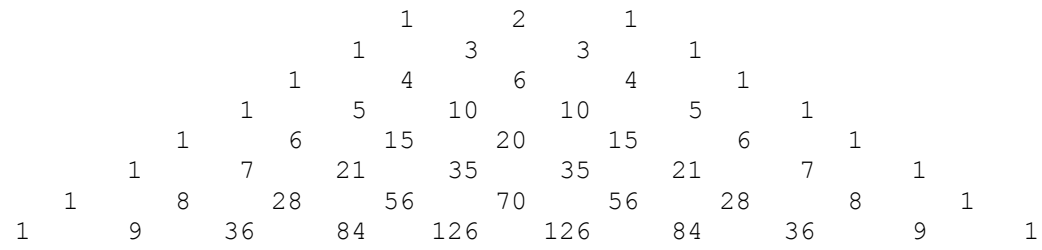
2.56. Pascal háromszög

2.56.1. Írjon programot, amely kiírja a képernyőre a Pascal háromszög első 10 sorát! Két szám között 4 szóköz karakter legyen! Az r-edik sor i-edik tagját számolja a következőképlettel:

```
r! / ( i! * ( r - i )! )
```

Kimenet:

```
                    1
                1       1
```



2.57. Lottó

2.57.1. Írjon lottó-generáló programot! A program generáljon 1 és 90 között 5 különböző pozitív egész számot! Egy fájl tippeket tartalmaz. Olvassa be a tippeket a fájlból, majd hasonlítsa össze őket a generált számokkal! Írja ki a képernyőre a generált nyerőszámokat, és azt, hogy hány embernek van 1-es, 2-es, 3-as, 4-es, illetve 5-ös találata a lottón!

Az input fájl formátuma:

Első sor: a tippek száma
Minden további sor 5 tippet tartalmaz, szóköz karakterekkel elválasztva.

Példa bemenet:

```
6
5 64 23 80 42
4 34 44 62 72
75 43 3 64 80
53 52 21 56 10
1 37 87 9 53
75 10 64 69 78
```

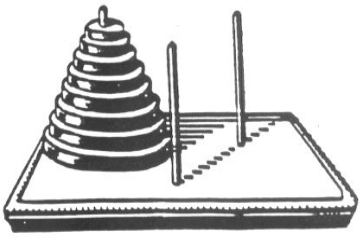
Kimenet:

```
Generated numbers: 55, 64, 68, 75, 50
There is 1 1 hit
There are 2 2 hits
There is 0 3 hit
There is 0 4 hit
There is 0 5 hit
```

2.58. Hanoi tornyai

2.58.1. A Hanoi tornyai játék 3 rudat tartalmaz (start, segéd, cél). Az első rúdon N db korong van egymás fölött, amelyek különböző méretűek. A korongok fentről lefelé növekvő sorrendben következnek, legnagyobb van legalul, a legkisebb legfelül, ahogyan az ábrán látható. A feladat a következő: a korongokat át kell rakni a start rúdról a célra, a segédrúd segítségével. Az átrakás szabálya, hogy nagyobb korongot kisebbre nem rakhatunk rá. Írjon programot ennek a feladatnak a szimulálására! A program argumentumként megkapja N értékét. Kezdetben minden korong a start rúdon van. Minden lépésben kérdezze meg, melyik rúdról melyikre tegyük át a legfelső korongot! Ha ez nem ellenkezik a szabályokkal, helyezzük át a korongot! Minden mozgatás után írja ki a képernyőre a korongok helyzetét, a példában látható módon! Ha a felhasználó 0-t gépel, vagy

minden korong a cél rúdon van, a program véget ér. Használjon dinamikus tömböket és struktúrákat a korongok helyének tárolására!



2.4. ábra: Hanoi tornyai szemléltetése

Példa bemenet:

```
************************************
Source: 3, 2, 1
Auxiliary: 0
Destination: 0
From: aux
To: src
Incorrect step!
************************************
Source: 3, 2, 1
Auxiliary: 0
Destination: 0
From: src
To: aux
************************************
Source: 3, 2
Auxiliary: 1
Destination: 0
From: src
To: dst
************************************
Source: 3
Auxiliary: 1
Destination: 2
From: aux
To: dst
************************************
Source: 3
Auxiliary: 0
Destination: 2, 1
From: 0
The disks are not on the destination!
************************************
Source: 3
Auxiliary: 0
Destination: 2, 1
```

2.59. Törtek összeadása

2.59.1. Egy fájl több törtet tartalmaz. Írjon programot, amely beolvassa a törteket, és összeadja őket! Minden összeadás után számolja ki az aktuális összeg számlálójának és nevezőjének legnagyobb közös osztóját az Euklideszi algoritmus segít-

ségével, és ha lehet, akkor egyszerűsítse a törtet, majd folytassa az összeadást! Minden műveletet írjon ki a képernyőre a példában látható módon! Az Euklideszi algoritmus megadja két szám legnagyobb közös osztóját.

```
procedure Euclidean(A, B)
  M = B
  while M is not zero {
    B = M
    M = A mod B       // M az A / B osztás maradéka
        A = B
}
```

Az input fájl formátuma:

Első sor: a törtek száma
A többi sorban két pozitív egész található, az első a számláló, a második a nevező értéke. A két érték szóköz karakterrel van elválasztva.

Példa bemenet:

```
3
2 3
4 5
6 11
```

Kimenet:

```
   2        4        22
------ + ------ =  ------
   3        5        15

  22        6       332
------ + ------ =  ------
  15       11       165

 332        4      1656        184
------ + ------ =  ------ =  ------
 165        3       495         55
```

2.60. Nevezetes számok

Számelmélettel már az ókori Görögországban is sokat foglalkoztak, például sok nevezetes tulajdonságú számot próbáltak meghatározni.

2.60.1. Írjon programot, amely megszámolja, hogy 1 és 10000 között mennyi tökéletes szám van! Tökéletes számnak tekintjük azokat a számokat, amelyekre igaz, hogy megegyeznek az osztóik összegével. Az osztók közé az 1-et is hozzávesszük, de magát a számot nem. Például tökéletes szám a 6, mert $1 + 2 + 3 = 6$.

2.60.2. Írjon programot, amely megszámolja, hogy 1 és 100 között mennyi hiányos szám található! Hiányos szám az, amely nagyobb az osztóinak összegénél, önmagát nem számítjuk az osztói közé. Például a 21 hiányos szám, mert $1 + 3 + 7 < 21$.

2.60.3. Írjon programot, amely megszámolja, hogy 1 és 1000 között mennyi bővelkedő szám található! Bővelkedő szám az, amely kisebb az osztóinak összegénél,

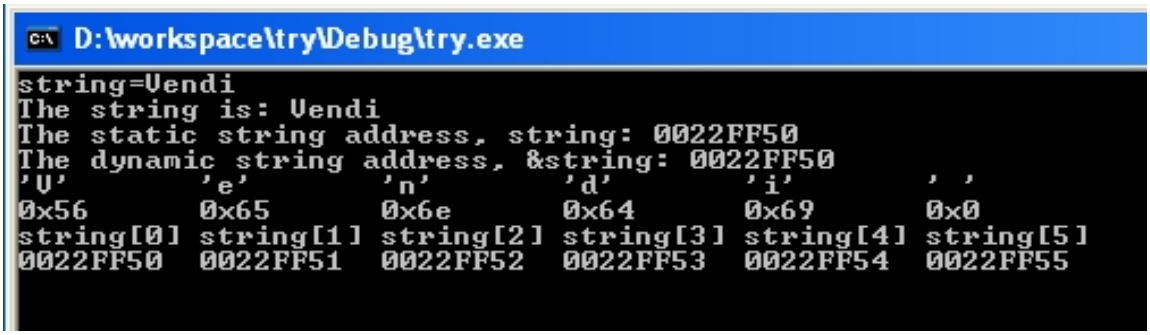
önmagát szintén nem számítjuk az osztói közé. Például a 12 bővelkedő szám, mert 1 + 2 + 3 + 4 + 6 > 12.

2.60.4. Módosítsa az előző feladatot úgy, hogy azon 1 és 1000 között található bővelkedő számokat adja meg, amelyek 2-vel vagy 3-al oszthatóak!

3. Tömbök és mátrixok feladatai

3.1. Egy sztring címei

3.1.1. Készítsünk programot, amely bekér egy legfeljebb 6 karakter hosszú sztringet és a karakteres képernyőn megjeleníti azt úgy, hogy kiírja az egyes karakterek értékeit, karakteresen és ASCII kóddal 16-os számrendszerben, valamint kiírja az egyes karakter változóknak a neveit és ezeknek a címeit. Ötlet mutatók kiírásához használja a %p-t!



3.1. ábra: Lehetséges képernyőkép

3.1.2. Alakítsa át a program kiírását úgy, hogy a bekért szó függőlegesen jelenjen meg és a hozzá tartozó kiírások az egyes karakterektől jobbra helyezkedjenek el!

3.2. Osztás és maradékképzés

3.2.1. Tároljon két tömbben 5-5 egész számot, az osztókat és az osztandókat! Az értékeket inicializálással adja át a tömböknek, legyen köztük negatívok is! Tippelje meg az egészosztás és maradékképzés eredményét! Írja ki a helyes választ és számolja, a helyes válaszok számát! Ötlet: ellenőrizze, hogy a második tömbben ne legyen nulla!

3.2.2. Továbbra is meg kell tippelni a megoldást, de az operandusokat generálja véletlenszerűen -10 és 10 között! Véletlenszerűen döntse el azt is, hogy egész vagy lebegőpontos osztást végez! Utóbbinál függvényt kell használni a maradékképzésre. Lebegőpontos osztásnál mindig írja ki a tizedes pontot az operandusoknál!

3.3. Sztring konvertálás

3.3.1. Készítsen programot, amely statikus karaktertömbben lévő 4 számot átkonvertál short int, int, float és double típusra! A számok összegét írja be az eredeti tömbbe!

3.3.2. A kezdeti sztringben 3 double típusú szám legyen, az elsőt eggyel, a másodikat kettővel, a harmadikat hárommal növeljük meg! A megnövelt értékeket egy új tömbbe írjuk ki normál alakban, 8 tizedes pontossággal és előjellel!

3.4. Sztring bekérés

3.4.1. Kérjen be egy-egy sztringet gets, getchar és scanf segítségével! Írja ki a sztringeket puts, putchar és printf segítségével!

3.4.2. Mi történik, ha a gets és a scanf részeket felcseréljük? Hogyan kezelhető a helyzet?

3.5. Gyökök

3.5.1. Írjon programot, amelyben egy 5 hosszú statikus float tömb minden elemébe a hozzátartozó index gyökét írja be! Használjon for ciklust az egyes elemek eléréséhez és az eredmény kiírásához! Az egyes értékek vesszővel legyenek elválasztva!

3.5.2. Oldja meg, hogy az utolsó szám után ne jelenjen meg vessző a kiírásnál és hogy a tömb nagyságát szabadon lehessen meghatározni!

3.6. Tömb reprezentálása

3.6.1. A karakteres képernyőn reprezentáljon egy 4 hosszú egész tömböt! Kérje be a tömb elemeit és írja ki az értékeket '*' szimbólumokból készített négyzetekbe! Adja meg az egyes elemek indexeit és címeit! Írja ki a tömb mint mutató értékét és annak a címét!

3.2. ábra: Lehetséges képernyőkép

3.6.2. Oldja meg ugyanezt a feladatot dinamikus tömbre is! Hol lesz eltérés?

3.7. Magasság mérés

3.7.1. Feladat: Írjon programot, amelyben egy 10*10-es statikus, inicializált, float értékeket tartalmazó mátrix tengerszint feletti magasság adatokat reprezentál! Adja meg a szárazföld és a tenger arányát, ha a negatív magasság tengerfeneket reprezentál! Adja meg a legmagasabb pontot és adja meg a legmagasabb pontot a tengerben feltételes maximum keresés segítségével! Azt az esetet is kezelni kell, amikor nincs tenger! Számolja ki a terület átlagos magasságát!

3.7.2. Határozza meg a legnagyobb szintkülönbséget a teljes területen, csak a szárazföldön, csak a tengerben!

3.8. Kockadobás

3.8.1. Írjon programot, amely kockadobást szimulál véletlen számok generálásával! Kérjen be egy sorozat hosszt, x-t, és addig generáljon véletlen számokat, amíg nem jelenik meg egymás után x darab hatos! Hány dobás után kaptuk meg a kívánt eredményt? Számolja az egyes dobások gyakoriságát!

3.8.2. Módosítsa úgy az előző programot, hogy egy x értékre 30-szor fusson le a program! Átlagosan hány dobás szükséges x darab egymást követő hatoshoz?

3.9. Csúsztatás

3.9.1. Írjon függvényt, amely 3 cím szerint átadott float paraméter értékét (a, b és c) elcsúsztatja a következő módon: ciklikus eltolás: a → b, b → c, c → a, nem ciklikus eltolás: a → b, b → c, 0 → a! Egy negyedik paraméter jelezze, hogy az eltolás ciklikus-e!

3.9.2. 3 változó helyett egy 3 elemű tömbön végezze el az elcsúsztatást!

3.9.3. Egy plusz változó jelezze, hogy melyik irányba történik a csúsztatás! A tömb mérete legyen megadható!

3.10. Műveletek tömbökön

3.10.1. Írjon függvényeket a következő feladatokra: határozza meg egy tömbben lévő elemek összegét, átlagát, minimumát, hogy egy adott elem szerepel-e benne! Egy fájlban legyen a main függvény és egy másikban a többi függvény!

3.10.2. Egészítse ki az előző programot a következő feladatokat ellátó függvényekkel! Maximális eleme indexének a meghatározása, szórás kiszámítása, két tömb elemeinek a felcserélése!

3.10.3. Egészítse ki az előző programot a következő feladatokat ellátó függvényekkel! Adott elem első előfordulásának meghatározása, utolsó előfordulásának meghatározása, két tömb között az alapműveletek elvégzése az azonos indexű elemek között, skaláris szorzat meghatározása.

3.11. Sztring átalakítás

3.11.1. Írjon programot, amely egy szöveg minden negyedik karakterét kihagyja, ha az nem white space! inicializálja úgy a sztringet, hogy abban több sor legyen és ezek a sorok a program kódban is sorokban legyenek (külön idézőjel párok felhasználásával)! A néggyel való oszthatóságot külön függvénnyel valósítsa meg!

3.11.2. Módosítsa úgy a programot, hogy a magánhangzókat hagyja ki, de csak akkor, ha nem egy szó elején vannak!

3.12. Szöveg statisztika

3.12.1. Készítsen magánhangzó statisztikát egy bekért sorról! A magánhangzó kis és nagy betűs alakját nem különböztetjük meg, csak az angol magánhangzókat tekintjük.

3.12.2. Készítsen statisztikát egy szövegben lévő szavak és mondatok számáról!

3.13. Kódolt beszéd

3.13.1. Írja meg a „tuvudsz ivigy beveszevelnivi” programot! A kimenet csupa nagybetű legyen! írja meg azt a függvényt, amely eldönti, hogy egy betű magánhangzó-e!

3.13.2. Bővítse úgy a programot, hogy az új sztring egy megfelelően hosszú dinamikus tömbbe kerüljön! Készítse el a visszafele kódoló függvényt!

3.14. Sztring kivonás

3.14.1. Írjon programot, amely egy sztringből kivon egy másikat! Például: almafavirág - fa = almavirág. Ha a második paraméter nincs az elsőben, akkor ne változzon az első paraméter érétke!

3.14.2. Módosítsa úgy a programot, hogy a második sztring minden előfordulását kiveszi az elsőből!

3.15. Kisbetű - nagybetű

3.15.1. Írjon programot, amely bekért mondatot formáz adott választás szerint! Menüből választhatóak a következő lehetőségek: UPPER CASE (mindent nagybetűre), lower case (mindent kisbetűre), tOGGLE CASE (kis betűket nagyra és nagyokat kicsire), Sentense case (a sor első betűje nagy csak), Title Case (minden szókezdő betű nagy). Mind a konvertálások, mind a menü függvénnyel legyen megoldva!

3.15.2. Valósítsa meg a „Nice Title Case” formázást! A szavak első betűje nagy kivéve az alábbi szavak esetén: and, of, the, a, an, from, for, in. A kivételt nem kell figyelembe venni a sor első karakterénél és egy kettős pont után. Például: „The Life of an English Man and the War”

3.16. Tömbnövelés

3.16.1. Írjon programot, amely bekéri egy tömb méretét, lefoglalja a helyet a tömb számára és feltölti elemekkel! Növelje meg a tömb méretét duplájára úgy, hogy létrehoz egy új tömböt, és belemásolja a régi tömböt! Töltse fel a maradék helyet is új értékekkel!

3.16.2. Alakítsa át úgy az előző programot, hogy függvényben kérje be tömb méretet és ott hozza azt létre!

3.17. Mátrixszorzás

3.17.1. Írjon programot két dinamikus mátrix összeszorzására! Függvényben foglaljon helyet a mátrixoknak, amelyeket véletlen egészekkel töltsön fel! Írjon függvényt a mátrixszorozásra, ellenőrizze, hogy a mátrix mértek kompatibilisek-e, a függvény visszatérési értéke legyen az új eredmény mátrix! Írassa ki az eredetei és az eredmény mátrixokat függvényben, majd szabadítsa fel azokat!

3.17.2. Készítse el az újraméretez függvényt, amely vagy levág a mátrixból egy darabot, és/vagy a mátrix jobb oldalához illetve aljához hozzáfűz egy új részt! Az új rész egységmátrix szerűen tartalmazzon 0-kat és 1-ket, ha az új rész nem négyzet alakú, akkor az egyesek fűrészfog szerűen helyezkedjenek el!

1			1			1
	1			1		
		1			1	

3.3. ábra: Téglalap alakú egységmátrix

3.18. Virtuális memória

3.18.1. Kérje be egy lebegőpontos tömb méretét! Ha az 5 vagy kisebb, akkor hozzon létre dinamikus tömböt a memóriában, ha nagyobb, akkor pedig bináris fájlt a háttértáron! Készítsen struktúrát, amely tartalmaz egy FILE*, double* és méret adattagot! Készítsen függvényeket: tároló inicializálására és felszabadítására, tároló adott pozíciójában történő olvasásra és írásra. Menüben kérje be, hogy írni vagy olvasni akar, a fő függvény számára a tárolás módja legyen transzparens!

3.19. Gépelés

3.19.1. Készítsen gépelést tanító programot! Sztring tömbben tároljon 20 különféle szót! Kérje be, hogy hány szót akar gépelni, véletlenszerűen válasszon ki egyet, amit megjelenít, és aztán várja azt bemenetként! Hibás bevitel esetén kérje be újból a szót! A gyakorlás végén írja ki az eltelt időt és a helyes és elrontott szavak számát!

3.20. Nagy számok összeadása

3.20.1. Írjon programot két nagyon nagy pozitív egész szám összeadására! A számok legfeljebb 100 jegyűek. Írja ki a képernyőre a két számot, majd az összegüket, a következő formátumban. A két számot fájlból olvassa be, a számok szóközzel vannak elválasztva. A számokat statikus karaktertömbökben tárolja!

Példa bemenet:

```
434234643643746 575434253245430
```

Kimenet:

```
434234643643746 + 575434253245430 = 1009668896889176
```

3.21. Medián

3.21.1. Írjon programot, amely számokat olvas be fájlból, majd kiszámítja ezek átlagát és mediánját! A program írja ki a képernyőre a számokat, az átlagot, és a mediánt is! A medián kiszámításához először sorba kell rendeznie a számokat. Páratlan számú adat esetén a medián ebben a sorban a középső elem, páros számú adat esetén a két középső elem átlaga.

Az input fájl formátuma:

Első sor: Az adatok száma.
Második sor: A számok, egy-egy szóköz karakterrel elválasztva.

Példa bemenet:

```
30
4 2 5 3 4 2 5 1 3 5 1 5 5 5 4 4 3 4 4 2 5 5 1 5 5 3 2 5 2 5
```

Kimenet:

```
30 numbers:
4 2 5 3 4 2 5 1 3 5 1 5 5 5 4 4 3 4 4 2 5 5 1 5 5 3 2 5 2 5
The average is 3.63333
The median is: 4
```

3.22. Ösztöndíj

3.22.1. Írjon programot, amely kiszámítja a hallgatók ösztöndíját a tanulmányi átlaguk alapján! Fájl tartalmazza a hallgatók adatait: neptun kódjukat, és a tárgyaik kreditértékét, illetve a szerzett érdemjegyeket. A program írja ki a képernyőre minden hallgató esetében a neptun kódot, a felvett tárgyak számát, a szerzett érdemjegyek kredittel súlyozott átlagát, a felvett és a teljesített kreditek összegét, és a hallgató számára megítélt ösztöndíjat! Egy tárgy teljesített, ha a hallgató átment az adott tárgyból. Az átlagba a felvett tárgyak és nem a teljesítettek tartoznak.

Az ösztöndíjak összege:

- 0 <= átlag < 2 → 0 HUF
- 2 <= átlag < 3 → 15 000 HUF
- 3 <= átlag < 4 → 20 000 HUF
- 4 <= átlag < 4.5 → 25 000 HUF
- 4.5 <= átlag <= 5 → 30 000 HUF

Az input fájl formátuma:

Első sor: A hallgatók száma.
A hallgatók adatait tartalmazó sor:
Sztring number1 // Sztring: a hallgató neptun kódja (6 karakter)
// number1: a felvett tárgyak száma
Az egyes tárgyakhoz tartozó sor:
number2 number3 // number2: a tárgy kreditértéke, number3: a tárgyból szerzett érdemjegy

Példa bemenet:

```
2
TGDS32 3
4 2
3 5
3 4
FBG4SW 3
4 5
2 3
4 1
```

A TGDS32 és FBG4SW neptun kódú hallgató adatait látjuk. TGDS32 3 tárgyat vett fel, az első tárgya 4 kredit értékű, az érdemjegye belőle 2.

Kimenet:

```
Neptun: TGDS32 3 subjects
Neptun: FBG4SW 3 subjects
TGDS32:
        Credits: 10/10
        Average: 3.5
        Bursary: 20000 HUF
FBG4SW:
        Credits: 10/6
        Average: 3
        Bursary: 20000 HUF
```

3.23. Szavak keresése

3.23.1. Írjon programot, amely beolvassa a „dune.txt” nevű fájlt és keresi benne az „Atreides” szót! A program írja ki a képernyőre az összes találatot az előtte lévő szóval együtt! Írja ki a találatok számát is! Ha egy írásjel szerepel az Atreides szó után, pl. „Atreides.”, „Atreides,”, az jó találatnak minősül, azonban amikor az Atreides szó egy részsztringje egy szónak, azt nem tekintjük jó találatnak, pl. „Atreides-Harkonnen”, „Atreides's”.

Az input fájl formátuma:

A szavak szóköz karakterekkel vannak elválasztva.

Példa bemenet:

```
Leto Atreides is the head of the Atreides house. Leto Atreides's son is
Paul.
```

Kimenet:

```
Leto Atreides
the Atreides
2 hits
```

3.24. Egyszerű sztringfordító

3.24.1. Írjon programot, amely beolvas egy fájlból utasításokat és végrehajtja a benne foglaltakat! Az utasítások három integer típusú változót használhatnak: A, B, C. A script elején a változók értéke 0. A lehetséges utasítások a következők:

GET Op: Op lehet egy változó (A, B, or C), ez a parancs beolvas egy tízes számrendszerbeli számot a billentyűzetről, és az Op-ban tárolja.
WRITE Op: Op értékét a képernyőre írja.
ADD Op1 Op2: Hozzáadja Op2-t Op1-hez, azaz Op1 = Op1 + Op2.
SUB Op1 Op2: Kivonja Op2-t Op1-ből, azaz Op1 = Op1 - Op2.
MUL Op1 Op2: Megszorozza Op1-et Op2-vel, azaz Op1 = Op1 * Op2.
DIV Op1 Op2: Elosztja Op1-et Op2-vel, azaz Op1 = Op1 / Op2.
STOP: Ez a parancs a script végét jelzi.

Minden sorban csak egy parancs szerepel. Ha a sor # karakterrel kezdődik, azt a sort megjegyzésnek tekintjük. Ezeknek a soroknak a végét is egy # karakter jelzi. Az első # után egy szóköz karakter következik. A program írja ki ezeket a sorokat a képernyőre, a # karakterek kivételével! Például ha a sor a következő „*# This is a comment! #*”, a program a következőt írja ki: „This is a comment!”

Példa bemenet:

```
# Type A: #
GET A
# Type B: #
GET B
ADD A B
# The sum: #
WRITE A
STOP
```

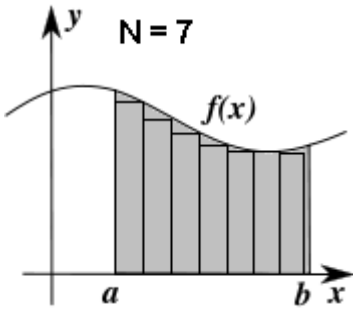
Kimenet:

```
Type A: 1
Type B: 2
The sum: 3
```

3.25. Riemann integrál

3.25.1. Írjon programot, amely négy függvény Riemann-integrálját számítja ki 0 és 1 között! Elsőként az [a, b] intervallumot felosztjuk N db (1000000) részre, azaz egy kis intervallum hossza: h = (b - a)/N. N téglalapot képzünk úgy, hogy egyik téglalap se lógjon a függvény fölé. Az i-dik téglalap alapja tehát h magassága pedig f(h*i) és f(h*(i + 1)) közül a kisebbik. Az integrál a téglalapok összege. A számolandó függvények:

f1(x) = $\sqrt{(2-x)*x}*4$
f2(x) = x^2
f3(x) = sin(x)
f4(x) = tan(x)



Ábra: Riemann integrál szemléltetése

Példa bemenet:

```
The Riemann integral of f1(x) over x from 0 to 1: 3.14159
The Riemann integral of f2(x) over x from 0 to 1: 0.333332
The Riemann integral of sin(x) over x from 0 to 1: 0.459696
The Riemann integral of tan(x) over x from 0 to 1: 0.615624
```

3.26. Polinomok összeadása

3.26.1. Írjon programot, amely polinomokat ad össze! Olvasson be fájlból két polinomot! Írja ki a képernyőre a két polinomot, illetve ezek összegét! A polinomok tárolására használjon dinamikus tömböket és struktúrákat.

Az input fájl formátuma:

Első sor: Number1 - az első polinom foka
Második sor: Number1 + 1 egész szám, az első polinom együtthatói (az utolsó szám a konstans).
Harmadik sor: Number2 - a második polinom foka
Negyedik sor: Number2 + 1 egész szám, a második polinom (az utolsó szám a konstans).
A számok egy-egy szóköz karakterrel vannak elválasztva

Példa bemenet:

```
5
-3 5 -4 0 4 2
```

```
4
5 4 5 -1 0
```

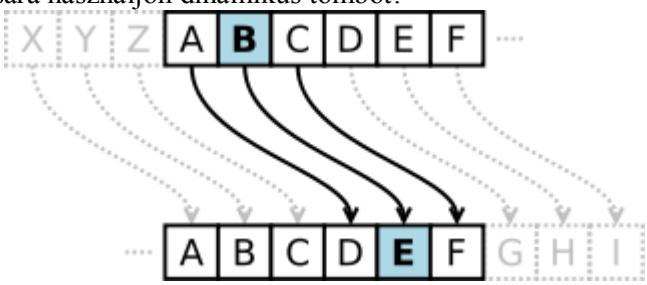
Kimenet:

```
 - 3x^5 + 5x^4 - 4x^3 + 4x + 2 +
5x^4 + 4x^3 + 5x^2 - x =
 - 3x^5 + 10x^4 + 5x^2 + 3x + 2
```

3.27. Caesar dekódoló

3.27.1. A Cézár-kódolás a következőképpen működik: Minden karaktert egy előre meghatározott N pozícióval eltolunk. Például, ha N = 1, az „APPLE” szó BQQMD lesz, ugyanis A-ból B lett, P-ből Q, stb. Ha N = 1, Z-ből A lesz. Példa N = 3-ra:

eredeti szöveg: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
kódolt szöveg: WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ
Az ellenségtől egy kódolt üzenetet kaptunk. Nem ismerjük N értékét, de a kémünk azt állítja, hogy az eredeti üzenet szavai közt szerepel a „THE” szó. Írjon programot, amely beolvassa a kódolt üzenetet a „codedmessage.txt” fájlból, és megfejti, mi volt az eredeti üzenet! N értéke 1 és 26 közt van, a programnak tehát meg kell vizsgálnia minden N-t, amíg az üzenet nem tartalmazza a „THE” szót. Az üzenet tárolására használjon dinamikus tömböt!



Ábra: Caesar kódolás szemléltetése

Az input fájl formátuma:

Első sor: Az üzenet szavainak száma.
Második sor: Az üzenet szavai, egy-egy szóköz karakterrel elválasztva.

Példa bemenet:

9
WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ

Kimenet:

```
The coded message: WKH TXLFN EURZQ IRA MXPSV RYHU WKH ODCB GRJ
The original message: THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
```

3.27.2. Legyen a tartalmazott szó, a mostani „THE” is bemenet! Ha több lehetséges visszakódolás van, akkor jelenítse meg az összest!

3.28. CD katalógus

3.28.1. Rendelkezik egy CD katalógussal, amit fájlban tárol. Írjon programot, amely képes arra, hogyha begépeli egy szoftver nevét, akkor megmondja, melyik CD-n van a szoftver! A szoftverek nevei maximum 15 karakter hosszúak, és minden CD-nek van egy neve,

amely szintén maximum 15 karakter hosszú. Használjon struktúrákat, illetve dinamikus tömböket a fájlból beolvasott adatok tárolására!

Az input fájl formátuma:

Első sor: A CD-k száma.
A további sorok:
String: A CD neve
Egész szám: A CD-n lévő szoftverek száma
Ezután a CD-n lévő szoftverek neve következik.

Példa bemenet:

```
2
2007/11
2
BurningStudio
RadioRama
 2005/4
3
Doc2PDF
Apollo
Stellarium
```

A példában Jamesnek 2 CD-je van, az első neve „2007/11”, ezen a CD-n két szoftver van: BurningStudio és RadioRama. A másik CD 3 szoftvert tartalmaz.

Kimenet:

```
Software name: Apollo
The Apollo is here: 2005/4
```

3.28.2. Írja át úgy az előző programot, hogy lehessen a keresésnél egy ? vagy egy * karaktert használni! A ? egy tetszőleges karakter helyett állhat, a * pedig a sztring végén lehet és azt jelezi, hogy még valamennyi karakter hátra van. Ha egy minta több program nevére is illeszkedik, akkor jelenítse meg mindet!

3.29. Leltár

3.29.1. Egy cég termékek vásárlásával és eladásával foglalkozik. Az év elején 10.000 Ft-ja van. Egy fájlban tárolja az információkat a termékekről, az eladásokról és a beszerzésekről. Írjon programot, amely a fájlból beolvassa az adatokat, és megmondja az egyes termékek mennyiségét, illetve a cég egyenlegét az eladások és beszerzések után! Használjon dinamikus tömböket és struktúrákat az adatok tárolására!

Az input fájl formátuma:

Első sor: Number1 - a termékek száma
A következő Number1 db sor:
String Number2 Number3 Number4
String: az i-edik termék neve
Number2: az i-edik termék mennyisége
Number3: az i-edik termék beszerzési ára
Number4: az i-edik termék eladási ára
Number5: a termékekkel kapcsolatos adás-vételek száma
Az ezután következő Number5 db sor:
Number6 Number7

Number6: Az aktuális termék indexe (nem a neve, hanem a sorszáma)
Number7: Ha ez a szám pozitív, a cég beszerzett a Number6-os termékből Number7 db-ot, beszerzési áron; ha negatív, akkor a cég a Number6-os termékből eladott Number7 abszolút értékének megfelelő számú darabot, eladási áron.

Példa bemenet:

```
2
hammer 100 500 800
screwdriver 50 300 400
2
1 10
2 -5
```

A példafájl esetében kétféle termékünk van: 100 db hammer és 50 db screwdriver. A cég 500 Ft-ért tud hammer-t beszerezni, és 800 Ft-ért tudja eladni. 2 féle termékmozgás történt, először vásároltak 500 Ft/db áron 10 db hammer-t, majd 400 Ft/db áron eladtak 5 db screwdriver-t.

Kimenet:

```
Inventory:
**************************************
hammer:
        Count: 100
        Purchase price: 500
        Shop price: 800
screwdriver:
        Count: 50
        Purchase price: 300
        Shop price: 400
Inventory:
**************************************
hammer:
        Count: 110
        Purchase price: 500
        Shop price: 800
screwdriver:
        Count: 45
        Purchase price: 300
        Shop price: 400
We have got 7000 HUF
```

3.30. Könyvtári rendszer

3.30.1. A könyvtárakban a könyvek az Egyetemes Tizedes Osztályozás (ETO, angolul UDC) szerint vannak osztályozva. Minden könyvnek van egy ETO száma, amit a könyv témája határoz meg. Az ETO szám első számjegye a fő témát határozza meg:

0. Általános művek, bibliográfia, könyvtárügy.
1. Filozófia, pszichológia, logika, etika.
2. Vallás, egyházak, teológia.
3. Társadalomtudományok, közigazgatás, jog, oktatás.
4. Nem használt
5. Matematika, természettudományok, fizika, kémia.
6. Alkalmazott tudományok, műszaki tudományok, orvostudományok.

7. Művészetek, játék, sport, szórakozás.
8. Nyelvészet, irodalom
9. Régészet, földrajz, életrajz, történelem

Például egy 543-as ETO számú könyvről tudjuk, hogy matematikával, vagy természet-tudományokkal foglalkozik. A többi számjegy a program szempontjából nem fontos. A „lib1.txt” fájlban könyvek adatai szerepelnek. A fájl tartalmazza a szerző nevét, a könyv címét, a kiadási évet és a könyv ETO számát. Írjon programot, amely beolvassa ezt a fájlt, listázza az egyes könyveket, végül összesítést ad, hogy hány könyv található a könyvtárban egy-egy a témában! Használjon dinamikus tömböket és struktúrákat az adatok tárolására!

Az input fájl formátuma:

Első sor: A könyvek száma.
A többi sor formátuma: Szerző Rövid_Cím Kiadási év ETO (szóköz karakterekkel elválasztva).
A szerző neve és a könyv címe maximum 20 karakterből áll, mindkettő egy-egy szó csak. Az ETO szám 3 karakterből áll.

Példa bemenet:

```
3
Asimov Foundation 1980 820
Herbert Dune 1990 820
Dancs Terror 2001 512
```

Kimenet:

```
Author: Asimov
Title: Foundation
Year of edition: 1980
UDC: 820 Language, Linguistics, Literature

Author: Herbert
Title: Dune
Year of edition: 1990
UDC: 820 Language, Linguistics, Literature

Author: Dancs
Title: Terror
Year of edition: 2001
UDC: 512 Matematika and natural sciences
**********************************
Generalities: 0 books
Philosophy, Psychology: 0 books
Religion, Theology: 0 books
Social sciences: 0 books
vacant: 0 books
Matematika and natural sciences: 1 books
Applied sciences, Medicine, Technology: 0 books
Arts, Recreation, Entertainment, Sport: 0 books
Language, Linguistics, Literature: 2 books
Geography, Biography, History: 0 books
```

3.31. Kikölcsönzött könyvek

3.31.1. A könyvtárak nyilvántartást vezetnek a kikölcsönzött könyvekről. Ez tartalmazza a könyvek szerzőjét, címét, illetve a kölcsönzés lejáratának idejét. Tételezzük fel, hogy az aktuális dátum 2008. december 15. Írjon programot, amely beolvassa a kikölcsönzött

könyvek listáját egy fájlból, és kilistázza azoknak a könyveknek az adatait, amelyek kölcsönzési ideje már lejárt! Használjon dinamikus tömböket és struktúrákat az adatok tárolására!

Az input fájl formátuma:

Első sor: A könyvek száma.
A többi sor: Szerző Rövid_cím Év Hónap Nap
A szerző neve és a könyv címe is egy-egy szó csak. Az év, hónap, nap a lejárat idejét jelölik.

Példa bemenet:

```
3
Asimov Foundation 2009 1 20
Herbert Dune 2008 12 1
Dancs Terror 2008 11 30
```

Kimenet:

```
Title: Herbert
Author: Dune
Expiraton: 2008 12 1

Title: Dancs
Author: Terror
Expiraton: 2008 11 30
```

3.32. Szótár

3.32.1. Írjon angol-magyar szótár programot! Fájl tartalmazza az angol-magyar szópárokat! A program olvasson be egy szót a billentyűzetről! A szó a két nyelv bármelyikén lehet. Ezután olvassa be a szópárokat a fájlból, és keresse meg a megadott szóhoz tartozó szópárt! Az is találatnak számít, ha a begépelt szó, az előtagja az egyik szótári bejegyzésnek. Írja ki a képernyőre a találatokat!

Az input fájl formátuma:

Első sor: A szópárok száma
A többi sor: Angol_szó Magyar_szó (szóköz karakterrel elválasztva)
Egy-egy szó maximális hossza 20 karakter.

Példa bemenet:

```
5
SUN NAP
DOG KUTYA
BICYCLE BICIKLI
VILLAGE FALU
SUNSHINE NAPSUTES
```

Kimenet:

```
Type the word: NAP
Hun->Eng: NAP SUN

1 hits
```

Kimenet:

```
Type the word: SUN
Eng->Hun: SUN NAP
Eng->Hun: SUNSHINE NAPSUTES

2 hits
```

3.33. Sudoku ellenőrző

3.33.1. Írjon programot, amely leellenőriz egy kitöltött Sudoku táblát! A tábla értékeit olvassa be fájlból! Egy tábla akkor van helyesen kitöltve, ha minden sorban, minden oszlopban és minden kis 3x3-as kis négyzetben az 1-9 közti számjegyek mindegyike egyszer szerepel. Az ábrán egy helyesen kitöltött tábla szerepel.

9	**5**	**8**	**2**	**7**	**4**	**6**	**3**	**1**
1	**4**	**7**	**5**	**3**	**6**	**9**	**2**	**8**
2	**6**	**3**	**9**	**8**	**1**	**4**	**7**	**5**
8	**3**	**1**	**4**	**6**	**9**	**2**	**5**	**7**
4	**2**	**5**	**3**	**1**	**7**	**8**	**9**	**6**
7	**9**	**6**	**8**	**5**	**2**	**1**	**4**	**3**
3	**1**	**4**	**7**	**2**	**8**	**5**	**6**	**9**
5	**8**	**9**	**6**	**4**	**3**	**7**	**1**	**2**
6	**7**	**2**	**1**	**9**	**5**	**3**	**8**	**4**

Ábra: Sudoku tábla

Az input fájl formátuma:

Az input fájl 9 sort és 9 oszlopot tartalmaz. A számjegyek egy-egy szóköz karakterrel vannak elválasztva.

Példa bemenet:

```
7 4 6 8 3 1 5 9 2
9 2 3 5 7 4 8 6 1
8 1 5 2 9 6 4 7 3
1 3 4 7 5 2 6 8 9
6 9 7 3 1 8 2 5 4
5 8 2 4 6 9 1 3 7
4 7 9 6 2 8 3 1 8
3 6 8 1 4 7 9 2 5
2 5 1 9 8 3 7 4 6
```

Kimenet:

```
This is a wrong table!
```

3.34. Amőba játék

3.34.1. Írjon egy egyszerű amőba játékot, 3x3-as pályára! Két játékos játszhat a programmal, akik a saját jelüket helyezhetik a mezőkbe felváltva (X vagy O). Az a játékos nyeri a

játszmát, akinek 3 jele lesz egy sorban, oszlopban vagy átlóban. Ha nincs több üres cella, vagy a játékosok valamelyike nyer, a játék véget ér. A programnak minden lépés után ki kell rajzolnia a játéktábla aktuális állását karakteresen! Feltételezzük, hogy mindkét játékos megfelelő pozíciót ad meg minden lépésben.

Példa bemenet:

```
   A B C
  -------
 1| | | |
  -------
 2| | | |
  -------
 3| | | |
  -------
First player
Target: A1
   A B C
  -------
 1|X| | |
  -------
 2| | | |
  -------
 3| | | |
  -------
Second player
Target: B1
   A B C
  -------
 1|X|O| |
  -------
 2| | | |
  -------
 3| | | |
  -------
```

```
First player
Target: A2
   A B C
  -------
 1|X|O| |
  -------
 2|X| | |
  -------
 3| | | |
  -------
Second player
Target: A3
   A B C
  -------
 1|X|O| |
  -------
 2|X| | |
  -------
 3|O| | |
  -------
First player
Target: B2
   A B C
  -------
 1|X|O| |
  -------
 2|X|X| |
  -------
 3|O| | |
  -------
```

```
Second player
Target: C1
   A B C
  -------
 1|X|O|O|
  -------
 2|X|X| |
  -------
 3|O| | |
  -------
First player
Target: C3
   A B C
  -------
 1|X|O|O|
  -------
 2|X|X| |
  -------
 3|O| |X|
  -------
First player won!
```

3.35. Térkép

3.35.1. Egy fájl egy magassági térképet tartalmaz. A magasságok pozitív egész számok. Írjon programot, amely beolvassa ezt a térképet egy mátrixba, kiírja a képernyőre és kiszámítja az adott területen az alföld, dombság, hegység és magashegység arányát! A mátrix legyen dinamikus! A magassági intervallumok a következők:

0 <= magasság < 250	alföld
250 <= magasság < 500	dombság
500 <= magasság < 1500	hegység
1500 <= magasság	magashegység

Az input fájl formátuma:

Első sor: number1 number2
A number1 a sorok, number2 az oszlopok számát adja meg.
Ezután number1 db sor következik, minden sorban number2 db, szóköz karakterrel elválasztott érték található.

Példa bemenet:

```
5 5
200 210 220 218 230
0 1600 322 31 1000
332 320 43 1 545
2000 32 252 0 321
1320 2123 324 21 43
```

Kimenet:

```
200  210  220  218  230
   0 1600  322   31 1000
 332  320   43    1  545
2000   32  252    0  321
1320 2123  324   21   43

Lowland: 52 %
Hill: 24 %
Mountain: 12 %
High mountain: 12 %
```

3.36. Inverz mátrix

3.36.1. Írjon programot, amely kiszámítja egy 3x3-as mátrix inverzét! A program írja ki a képernyőre a mátrix adjungáltját, determinánst és az inverz mátrixot. Az inverz mátrixot a következőképpen számíthatjuk ki: A^{-1} = adj(A) / det(A), ahol adj(A) az A mátrix adjungáltja, és det(A) az A mátrix determinánsa. Ha det(A) = 0, akkor A nem invertálható.

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

Ábra: Mátrix elemeinek az indexelése

Determináns:

A 2x2-es mátrix determinánsa a következő: a11*a22 - a21*a12
A 3x3-as mátrix determinánsa: a11 * a22 * a33 + a12 * a23 * a31 + a13 * a21 * a32 - a13 * a22 * a31 - a12 * a21 * a33 - a11 * a23 * a32

Adjungált:

$$\mathrm{adj}(\mathbf{A}) = \begin{pmatrix} +\begin{vmatrix} A_{22} & A_{23} \\ A_{32} & A_{33} \end{vmatrix} & -\begin{vmatrix} A_{12} & A_{13} \\ A_{32} & A_{33} \end{vmatrix} & +\begin{vmatrix} A_{12} & A_{13} \\ A_{22} & A_{23} \end{vmatrix} \\ -\begin{vmatrix} A_{21} & A_{23} \\ A_{31} & A_{33} \end{vmatrix} & +\begin{vmatrix} A_{11} & A_{13} \\ A_{31} & A_{33} \end{vmatrix} & -\begin{vmatrix} A_{11} & A_{13} \\ A_{21} & A_{23} \end{vmatrix} \\ +\begin{vmatrix} A_{21} & A_{22} \\ A_{31} & A_{32} \end{vmatrix} & -\begin{vmatrix} A_{11} & A_{12} \\ A_{31} & A_{32} \end{vmatrix} & +\begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \end{pmatrix}$$

ahol

$$\begin{vmatrix} A_{ij} & A_{kl} \\ A_{mn} & A_{op} \end{vmatrix} = \det\begin{pmatrix} A_{ij} & A_{kl} \\ A_{mn} & A_{op} \end{pmatrix};$$

Példa bemenet:

```
Original matrix:

 1  2  3
 2  4  5
 3  5  6

Adjugate matrix:

-1  3 -2
 3 -3  1
-2  1  0

Determinant of the matrix: -1
Inverse matrix:

 1 -3  2
-3  3 -1
 2 -1 -0
```

3.37. Mátrixműveletek

3.37.1. Írjon programot mátrixösszeadás és skalárral való szorzás megvalósítására! A program olvassa be fájlból két mátrixot! Adja össze a két mátrixot, és az eredményt szorozza meg 2-vel! A mátrixokat dinamikusan hozza létre! Ha A, B és C azonos méretű mátrixok, akkor $C_{ij} = A_{ij} + B_{ij}$, ahol C az A és B mátrixok összege. Egy mátrix „d” skalárral való szorzásakor a mátrix minden elemét „d”-vel szorozzuk.

Az input fájl formátuma:

Első sor: Két szám, az első a sorok, a második az oszlopok száma.
A többi sorban a két mátrix található, a példában látható módon. Minden sorban egy-egy mátrix-sor van, a számok szóköz karakterekkel vannak elválasztva. A mátrixok elemei egész számok.

Példa bemenet:

```
2 3
4 3 4
3 1 5

2 4 1
1 7 3
```

Kimenet:

```
A :
  4 3 4
  3 1 5

B :
  2 4 1
  1 7 3

************************
 (A + B) * 2 =
 12 14 10
  8 16 16
```

3.38. Morze kód

3.38.1. Írjon programot, ami egy üzenetet Morze kóddá alakít! Az üzenet begépelése után a program foglaljon le memóriát a kódolt üzenet számára, kódolja az üzenetet és jelenítse meg azt! Használja az alábbi sztring tömböt az egye karakterek leképezésére! A szóköz karakter kódja 5 pont.

```
char* morsecodes[26] = { ".=", "=...", "=.=.",     // A, B, C
                         "=..", ".", "..=.",       // D, E, F
                         "==.", "....", "..",      // G, H, I
                         ".===", "=.=", ".=..",    // J, K, L
                         "==", "=.", "===",        // M, N, O
                         ".==.", "==.=", ".=.",    // P, Q, R
                         "...", "=", "..=",        // S, T, U
                         "...=", ".==", "=..=",    // V, W, X
                         "=.==", "==.."};          // Y, Z
```

Példa bemenet:

```
Az üzenet: SOS I AM WRITING A TEST
Üzenet: "SOS I AM WRITING A TEST"
A kódolt üzenet:
"...===.................===......==.=...=..=.==.......=.....=....="
```

3.39. Mátrix szorzása vektorral

3.39.1. Írjon programot, amely megszoroz egy mátrixot egy vektorral! A mátrixot és a vektort fájlból olvassa be és tárolja őket dinamikus adatszerkezetekkel! Írja ki a képernyőre az eredeti mátrixot és vektort, majd a szorzatvektort! A szorzatvektor annyi értéket tartalmaz, amennyi a mátrix sorainak száma. Ha „P” a szorzatvektor, „A” a mátrix, és „V” az eredeti vektor, akkor
P_i = Sum(j=1…c, A_{ij}*V_j), ahol A_{ij} a mátrix i-edik sorának j-edik oszlopában van, c a mátrix oszlopainak száma.

Az input fájl formátuma:

Első sor: num1 num2
num1: a mátrix sorainak száma
num2: a mátrix oszlopainak száma, amely egyben az eredeti vektor elemeinek száma is
A következő num1 db sor a mátrix sorait tartalmazza, mindegyik sorban num2 db érték található, szóköz karakterrel elválasztva.
A mátrix után num2 db szám következik, az eredeti vektor elemei.

Példa bemenet:

```
2 3
4 1 3
7 2 2
6
2
3
```

Kimenet:

```
Matrix :
  4   1   3
  7   2   2
Vector :
 6
```

```
 2
 3
**********************
 Matrix * Vector =
 35
 52
```

3.40. Sztring tokenizáló

3.40.1. Írjon sztring tokenizáló programot! A felhasználó egy sztringet gépel be, amely több szóból állhat, a szavakat egy vagy több szóköz karakter választja el. Ezután a program számolja meg a sztringben lévő szavakat, foglaljon le dinamikus tömböket a szavak számára, és másolja be az egyes szavakat a lefoglalt karaktertömbökbe! Írja ki a program a képernyőre az eredeti sztringet, és az egyes szavakat!

Példa bemenet:

```
This is a           simple example.
The typed text: "This is a           simple example."
1.: "This"
2.: "is"
3.: "a"
4.: "simple"
5.: "example."
```

3.41. Szavak kicserélése

3.41.1. Egy fájlban egy legfeljebb 1000 karakter hosszú sztring található. Írjon programot, amely beolvassa ezt a fájlt, majd bekér a felhasználótól két szót! A második szó nem lehet hosszabb az elsőnél. A program keresse meg az első szó minden előfordulását a szövegben, és cserélje ki a második szóra. Írja ki a képernyőre az új szöveget, és a találatok számát!

Példa:

```
The original text: "This is a very simple text that can help for you to
understand the task. "

Type a word: simple
Type the new word: easy
1 hits

The new text: "This is a very easy text that can help for you to
understand the task. "
```

3.42. Ellenőrző összeg

3.42.1. Írjon programot, amely képes ellenőrző összeg generálására egy üzenethez, és ezen összeg alapján ellenőrizni tudja a bejövő üzeneteket! Az ellenőrző összeg kiszámítása a következőképpen történik: Adjuk össze az üzenet minden bájtját, ha az összeg nagyobb, mint 255, akkor az összegnek csak a legkisebb bájtját tekintjük. Az így kapott bájt kettes komplemense (255 - sum + 1) lesz az ellenőrző összeg. Például az „ABCD” üzenet bájtjai: 0x41, 0x42, 0x43 and 0x44, így az összeg = 0x41 + 0x42 + 0x43 + 0x44 = 0x10A. Ebből legkisebb bájt: 0xA, a kettes komplemens pedig: 0xFF - 0xA + 0x1 = 0xF6. Az ellenőrzés menete: Az üzenettel együtt az ellenőrző összeg is megérkezik egy

fájlban. A programnak össze kell adnia a beérkezett üzenet minden bájtját és az ellenőrző összeget. Ha az eredmény utolsó bájtja 0, akkor a beérkezett üzenet valószínűleg nem sérült. Egy fájl több üzenetet tartalmaz, az ellenőrző összegükkel együtt. A program olvassa be ezeket, és ellenőrizze őket! Írja ki a képernyőre az üzeneteket, az ellenőrző összegeket, és azt, hogy az üzenetek hibásak vagy helyesek! Végül számolja ki a „This is a simple checksum example.” üzenet ellenőrző összegét!

Az input fájl formátuma:

Első sor: Az üzenetek száma
A többi sor felváltva tartalmaz üzeneteket és ellenőrző összegeket. Az üzenetek soraiban több szó is szerepelhet. Az üzenet után következő sor az üzenet ellenőrző összegét tartalmazza hexadecimális formában.

Példa bemenet:

```
2
This is the first row of the message.
C2
This is the second row ov the message.
6E
```

Kimenet:

```
"This is the first row of the message."  C2 ===> Correct!
"This is the second row ov the message."  6E ===> Faulty!

Sum of "This is a simple checksum example.": C6C
The checksum: 94
```

3.43. Statisztika

3.43.1. Egy 5000 fős település lakosainak adatait kell feldolgoznunk. Van egy fájlunk, amely a lakosok életkorát tartalmazza. Írjon programot, amely kiírja a képernyőre, hogy mennyi kiskorú, felnőtt, nyugdíjas él a teleülésen! Kiskorúnak számít az, aki még nem töltötte be a 18. életévét, egyébként a felnőttek közé soroljuk. A nyugdíjkorhatár 62 év.

3.43.2. Bővítse ki az előző programot úgy, hogy a program kiírja, hogy hányan születtek a második világháború alatt (1939-1945). Az aktuális év 2010.

3.43.3. Bővítse ki az előző programot úgy, hogy a program kiírja azt is, hogy hányan születtek szökőévben! Szökőévnek számítanak azok az évek, ahol az évszám osztható 4-el, de a 100-asra végződőek közül csak azok, amelyek oszthatóak 400-al. Ez alapján szökőév volt 1992, 1996, de 1900 nem, viszont 2000 igen.

3.44. Kerítés

3.44.1. Egy 400 méter kerületű telket szeretnénk körülvenni drótkerítéssel. Több köteg drótkerítésünk van, amelyek hosszát egyenként ismerjük. Írjon programot, amely 0 végjelig beolvassa az egyes kötegek hosszait, majd a végén kiírja, hogy összesen hány méternyi kerítésünk van és, ha kell-e még kerítést vennünk, akkor összesen mennyit!

3.45. Jegyek átlaga

Aladár év végén szeretné kiszámolni, hogy mely tantárgyból hányasra áll. Minden tárgyból sok jegyet szerzett, ezért úgy véli, fárasztó munka lenne kézzel átlagokat számolni, ezért számítógépes segítséghez folyamodik.

3.45.1. Írjon programot Aladárnak, amely beolvas 10 jegyet, majd kiszámolja azok számtani átlagát!

3.45.2. Bővítse ki az előző programot úgy, hogy 0 végjelig olvassa be a jegyeket és úgy végzi el a számolást!

3.45.3. Fejlessze tovább a programot úgy, hogy a jegyek szórását is meghatározza! Határozza meg az egyes jegyek átlagtól való eltérésit, utána ezen eltérések négyezeteinek az átlagát, majd az így kapott eredmény gyökét!

3.46. Nyúltenyésztés

3.46.1. Mr. Fibonacci elhatározta, hogy belevág a nyúltenyésztésbe. Az első hónapban vesz egy újszülött nyúlpárt. Minden nyúl 2 hónap után válik termékennyé. Minden hónapban minden termékeny nyúlpár egy új nyúlpárt szül. A nyulakat etetni kell, minden nyúl egy zsák nyúltápot fogyaszt el minden hónapban és egy nyúlpár se pusztul el. Írjon programot, amely meghatározza, hogy hány zsák nyúltápot kell vásárolni egy évre, ha az első hónapban 1 nyúlpár van!

3.46.2. Bővítse ki az előző programot úgy, hogy 3 évre előre számoljon, figyelembe véve, hogy Mr. Fibonacci minden év elején eladja a termékeny nyulak 90 %-át, és a terméketlenek 95 %-át! Amennyiben az eladandó nyulak mennyisége nem egész szám, úgy ezt az értéket mindig lefele kerekítjük. Például 101 termékeny nyúl esetén 90-et adunk el.

3.47. Jegyek

3.47.1. Egy 30 fős osztályban minden tanulóról tudjuk, hogy informatikából mennyi a jegyeinek az átlaga. Írjon programot, amely billentyűzetről beolvassa a 30 átlagot, majd kiírja a képernyőre, hogy volt-e bukás idén! Az bukik meg, akinek az átlaga kevesebb, mint 1.5.

3.47.2. Módosítsa az előző programot úgy, hogy addig olvassa be az átlagokat, amíg bukott embert nem talál! Amennyiben 0-t olvasunk be, az azt jelenti, hogy vége a beolvasásnak, vagyis ez nem egy átlag. A program ez esetben is írja ki, hogy bukik-e valaki!

3.48. Marsjáró

A Mars Explorer 3000-es marsjáró robot utazása során 100 méterenként feljegyzi, hogy a bolygón való leszállóhelyéhez viszonyítva milyen magasan / mélyen jár, így egy domborzati keresztmetszetet továbbít a földi irányító központba. Szeretnénk részletesen megvizsgálni a bejárt terepet. A vizsgálatra számítógépet használunk. A domborzati magasságokat egy tömbben tároljuk.

3.48.1. Írjon programot, amely a tömbben tárolt domborzati adatokból meghatározza, hogy mekkora volt a legmagasabb, illetve legalacsonyabb magasság, ahol a marsjáró járt, valamint hogy hol voltak ezek a pontok!

3.48.2. Közeledik a marsbeli tél, ezért a marsjáró földi vezetői attól félnek, hogy azelőtt leáll, mielőtt elküldené a legújabb összegyűjtött méréseket. Ezért beállítják, hogy minden egyes mérés után küldje el a Földre az aktuális magasságát. Írjon programot, amibe a kezelő személy begépeli az aktuális kapott magasság értéket és minden új adat után kiszámolja, hogy eddig mekkora volt a legmagasabb és legalacsonyabb pont és hol voltak azok!

3.48.3. Bővítse ki az előző programot úgy, hogy azt is nyilván tartsa a legmeredekebb lejtő és emelkedő meredekségét, valamint pozícióját! Megjegyzés: két mérési adat közötti különbség egy derékszögű háromszög egyik befogóját határozza meg, a másik befogó 100 méter.

3.48.4. Bővítse ki az előző programot úgy, hogy a leghosszabb sík terület hosszát, valamint kezdő és vég pontjának pozícióját keresse meg! Sík területnek számít az, ahol a terep pontjai összefüggően azonos magasságúak.

3.49. Ritka vektor

3.49.1. Nemo kapitány alámerül a Nautilus tengeralattjáróval, hogy megvizsgálja a tenger élővilágát. Merülés közben a segédei minden tízedik méternél megszámolják, hogy mennyi halat látnak, hogy ezt felhasználhassák későbbi kutatásaikhoz. A számlálások eredményét egy vektorban tárolják el, ahol az i. komponens a 10*i méter mélyen végzett számlálás eredményét tárolja. Sajnos még nem áll rendelkezésre nagy memória kapacitású számítógép, az eredményeket lyukkártyára írják. Megfigyelték, hogy nagyon gyakran nem látnak egy halat sem, ezért úgy gondolták, oly módon tárolják az eredményeket, hogy ezt a tényt kihasználják és a ritka vektorok tárolási módszeréhez folyamodtak, így helyet takarítottak meg, és az adatok feldolgozása is gyorsabb. Írjon programot, amely egy ilyen ritka vektorokat kezel! A ritka vektorok olyan vektorok, amelyek nagyon kevés nem 0 komponenst tartalmaznak. Ezeket célszerű úgy tárolni, hogy csak a nem 0 elemeket tároljuk, azon vektoron belüli indexével. Ennek egyik módja, ha a (szám, index) párokat láncolt listában tároljuk. A vektor dimenziója nincs korlátozva. Valósítsa meg a következő műveleteket:

- Beállít(index, érték): A vektor adott indexű elemét az adott értékre állítja be. Ha az érték 0, akkor az elem nem tárolódik.
- Érték(index): A művelet visszaadja az adott indexű elem értékét.
- Töröl: A művelet nulláza a teljes vektor tartalmát.
- NemNullák: A művelet visszaadja, hogy mennyi nem 0 elem van a vektorban.
- Legutolsó: A művelet visszaadja, hogy hányas indexen található a legutolsó nem 0 elem.
- Megjelenít: A művelet kiírja a képernyőre a teljes vektort egészen a „Legutolsó” által meghatározott elemig. A 0-kat is meg kell jeleníteni.

3.49.2. Előfordulhat, hogy az előző vektoron belül az elemek nem jó sorrendben követik egymást. Például lehet, hogy előbb található meg az 5-ös indexű elem a listában, mint a 3-as. Implementálja a „Rendez” műveletet, amely az indexeknek megfelelően növekvő sorrendbe rendezi a listán belüli elemeket!

3.49.3. Bővítse ki az előző programot az összeadás és a szorzás műveletekkel! Az összeadás két ritka vektort ad össze. Ügyeljen arra, hogy a 0 elemeket ne tárolja el! A szorzás 2 ritka vektort szoroz össze skalárisan.

4. Láncolt listák feladatai

4.1. Lista két tömbbel

4.1.1. Készítsen programot, amelyben egy (nem ciklikus) láncolt listát két tömbbel reprezentál! Az egyik tömb az elemek értékeit, a másik a következő elem indexét tartalmazza. Egész változó tartalmazza a lista első elemének az indexét! Inicializálja a tömböket a következő értékekkel: értékek: {34, 12.55, 893.2, 2, 11.6, 47.5, 45.3}, indexek: {-1, 0, 4, 1, 6, 3, 5}! A kettes indexű elem a lista első tagja. Írjon függvényt a lista elemeinek listázáshoz! Írjon függvényt az utolsó elem törléséhez, törölje egyenként az elemeket és minden törlés után listázza az elemeket! Hogyan kell módosítani a programot, ha törlés után elem beszúrása következik?

4.1.2. Írjon függvényt, amely a lista első elemét törli! Írjon függvényt, amely ellenőrzi, hogy az tömb konzisztens-e, vagyis a benne lévő indexek a megengedett tartományban vannak-e, a start elemtől indulva nem jelentkezik kör!

4.1.3. Valósítsa meg hasonló módon a ciklikus listát! Listázáskor lehessen megadni, hogy honnan kezdődjék a kiírás!

4.2. Lista egy tömbbel

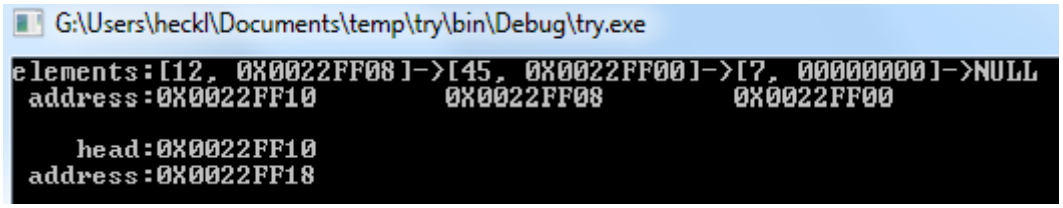
4.2.1. Készítsen programot, amelyben egy (nem ciklikus) láncolt listát statikus tömbbel reprezentál, úgy, hogy a tömb rekordokat tartalmaz, amelyekben név, életkor és a következő elem indexe adattag szerepel! írjon függvényt, amely a lista elejére fűz be új elemet! Az új rekord helyét a tömbben véletlen szám generátorral határozza meg a következő módon: létrehoz egy véletlen indexet; ha az azon indexhez tartozó rekord következő elem adattag értéke -2, akkor a kérdéses rekord nem tartalmaz hasznos információt vagyis nem használt rekordról van szó; egyébként nézzünk meg egy másik véletlen indexű elemet! (A módszer hátránya, hogy ha nincs több szabad hely, akkor a foglalás végtelen ciklusba kerül, a korrekt megoldás az lenne, ha külön tartanánk nyílván a szabad helyeket.)

4.2.2. Írjon függvényt a lista kiíratásához! Egyenként adjon elemeket a listához és írja ki az eredmény listát!

4.2.3. Módosítsa úgy az előző programot, hogy jelezze, ha nem sikeres az elem hozzáadása, mert betelt a tároló tömb! Dinamikus tömbben tárolja a listát, ha tömb betelne, akkor foglaljon le egy nagyobb tömböt! Valósítsa meg a lista végére beszúró függvényt!

4.3. Lista szerkezet

4.3.1. A karakteres képernyőn jelenítsen meg egy 3 hosszú, egyszeresen láncolt listát! A listát közvetlen értékadásokkal hozza létre, jelenítse meg a listát reprezentáló fej változót is!



4.1. ábra: Lehetséges képernyőkép

4.3.2. Hasonló módon jelenítse meg az oda-vissza láncolt listát!

4.4. Láncolt lista dinamikus rekordokkal

4.4.1. Írjon programot, amely dinamikus struktúrákkal valósítja meg az egyszeresen láncolt listát! Minden rekord tárolja a következő adatokat egy motorról: név, ár, teljesítmény! Valósítsa meg a beszúrást az x. elem után, a listázást és a listatörlést függvényekkel! Ha nincs x darab eleme a listának, akkor a lista végére szúrjon be! Őrszemmel tárolja a lista fejét! Adjon 4 elemet a listához, mindig a második elem után!

4.4.2. Készítsen függvényt adott sorszámú elem törlésére!

4.4.3. Írjon függvényeket, amelyek adott mezők alapján keresnek meg elemeket! Legyen olyan változat, amely megadja az első előfordulást, adott indexű elemtől tekintve az első előfordulást, az összes előfordulást! Írjon függvényt, amely sorszám alapján felcserél két elemet, adott tulajdonságú elemeket töröl ki, fordítva járja be a listát (és írja ki az elemeit)!

4.5. Sablon

4.5.1. Készítsen egy repülőgép osztályt azonosító, utasok, előző, következő adattagokkal! Készítsen kétszeresen láncolt listát, sablon osztállyal! Tárolja a lista elejét, végét és az aktuális elemet! Valósítsa meg a következő metódusokat: konstruktor, destruktor, adott elem keresése, elem beszúrása aktuális elem elé, aktuális elem törlése, aktuális mutató mozgatása relatív indexszel, aktuális elem referenciájának visszaadása, elemszám visszaadása, kiíró operátor. Tesztelje a listát a repülőgép osztály felhasználásával!

4.5.2. Írjon metódust az aktuális mutató abszolút mozgatására, adott sorszámú elem törlésére, adott objektum törlésére, a teljes lista törlésére, aktuális mutató utáni beszúrásra!

4.6. Lista

4.6.1. Írjon programot, amely számokat olvas be a billentyűzetről addig, amíg a beolvasott szám nem 0! A program fűzze a beolvasott számot egy egyszeresen láncolt lista végére! A -1 gépelése után írja ki a képernyőre a lista tartalmát! Valósítsa meg az add és print függvényeket, valamint a lista felszabadítását!

Példa:

```
Type a number: 10
Type a number: 20
Type a number: 30
Type a number: -1
The list: 10 20 30
```

```
Type a number: 40
Type a number: 50
Type a number: -1
The list: 10 20 30 40 50
Type a number: 0
```

4.7. Prímszita

4.7.1. Kérjünk be egy tetszőleges egész számot (n), majd egy listába tegyük be 2-től n-ig a természetes számokat! Vegyünk ki minden nem prím számot a listából! Ötlet: a lista első eleme (2) prím, minden olyan számot törlünk, amely osztható ezzel a számmal (az első iterációban 2 kivételével az összes páros szám törlődik). A megmaradt elemek közül a következő ismét prím, töröljük ennek is minden többszörösét. Ezt ismételjük addig, amíg a lista végére nem érünk, ekkor írjuk ki a listát!

4.8. Halmazműveletek

4.8.1. Hozzunk létre két rendezett listát, majd készítsük el a két lista unióját, az eredményt az első listában tároljuk!

4.8.2. Unió helyett a két lista metszetét képezzük!

5. A verem és sor adatszerkezetek feladatai

5.1. Verem osztály

5.1.1. Készítsen egy verem osztályt, amely statikus, 5 nagyságú egész tömbben tárol értékeket! valósítsa meg a következő metódusokat: init, push, pop, isFull, isEmpty! Ötlet: egyértelműen definiálja, hogy a verem mutató az első szabad elemre vagy az utolsó használt elemre mutat-e!

5.1.2. Valósítsa meg a vermet dinamikus tömbbel! Ha betelt a tömb, foglaljon le egy nagyobbat

5.2. Verem

5.2.1. Írjon programot, amely számokat olvas be a billentyűzetről addig, amíg a beolvasott szám nem 0! Tárolja a számokat egy veremben, amit láncolt listával valósítson meg. A 0 gépelése után a program olvassa ki a számokat a veremből, és írja a képernyőre! Valósítsa meg a push és pop függvényeket, valamint a lista felszabadítását!

Példa:

```
Type a number: 1
Type a number: 5
Type a number: 7
Type a number: 0
7
5
1
```

5.3. Sor

5.3.1. Írjon programot, amely számokat olvas be a billentyűzetről addig, amíg a beolvasott szám nem 0! Tárolja ezeket egy sorban, amit láncolt listával valósíts meg! A 0 gépelése után a program olvassa ki a számokat a sorból, és írja a képernyőre! Valósítsa meg a push és pop függvényeket, valamint a lista felszabadítását!

Példa:

```
Type a number: 10
Type a number: 43
Type a number: 2
Type a number: 5
Type a number: 0
10
43
2
5
```

5.4. Fordított lengyelforma

5.4.1. Egy matematikai kifejezést felírhatunk úgynevezett fordított lengyelformában is. Például a 3 + 4 kifejezés ebben az alakban így néz ki: 3 4 +. Fordított lengyelformában nincs szükség zárójelekre, és a műveletek precedenciájára sem kell ügyelni, mert azt már figyelembe vettük akkor, amikor meghatároztuk ezt a formát. A fordított

lengyelformában lévő kifejezés egy verem segítségével értékelhető ki. Balról jobbra olvassuk a kifejezést, ha szám következik, akkor azt berakjuk a verembe, ha műveleti jel, akkor kiveszünk a veremből 2 elemet, elvégezzük rajtuk a műveletet, majd az eredményt visszahelyezzük a verembe. A (2 + 5) * 3 - 9 / (2 + 1) kifejezés fordított lengyelformában: 2 5 + 3 * 9 2 1 + / -. Az algoritmus az alábbi táblázatban lévő lépéseket hajtja végre a kifejezés kiértékeléséhez. Írjon programot, amely beolvas egy matematikai kifejezést fordított lengyel formában, majd az ismertetett módszer szerint megoldja azt!

Lépés	Verem tartalma	Kifejezés	Művelet
0		2 5 + 3 * 9 2 1 + / -	
1	2	5 + 3 * 9 2 1 + / -	Verembe(2)
2	2 5	+ 3 * 9 2 1 + / -	Verembe(5) a = Veremből
3	7	3 * 9 2 1 + / -	b = Veremből Verembe (b + a)
4	7 3	* 9 2 1 + / -	Verembe(3) a = Veremből
5	21	9 2 1 + / -	b = Veremből Verembe(b*a)
6	21 9	2 1 + / -	Verembe(9)
7	21 9 2	1 + / -	Verembe(2)
8	21 9 2 1	+ / -	Verembe(1) a = Veremből
9	21 9 3	/ -	b = Veremből Verembe(b+a) a = Veremből
10	21 3	-	b = Veremből Verembe(b/a) a = Veremből
11	18		b = Veremből Verembe(b-a)

5.5. Várólista

5.5.1. Egy fővárosi hivatalban az ügyfeleknek rendszerint várakozni kell. Minden újonnan érkező személy kap egy azonosítót és az embereket ezen azonosítók alapján szólítják. A kiszolgálás érkezési sorrendben történik. Írjon programot, amely segít a hivatal személyzetének a következő kiszolgálandó ügyfél azonosítójának megállapításában! Valósítsa meg a sor adatszerkezetet, ahol a beszúrás művelet létrehoz egy új azonosítót, kiírja a képernyőre és hozzáadja a sorhoz! A kivesz művelet kiveszi a sorból egy elemet és az azonosítót kiírja a képernyőre. Menü segítségével lehessen választani a beszúrás és a kivesz műveletek között!

5.5.2. Előfordulhat, hogy egyszerre több embert tudnak kiszolgálni. Bővítse ki a programot az N_kivesz művelettel, amely N darab elemet vesz ki a sorból és kiírja a megfelelő azonosítókat!

5.6. Nyomtatók

A fővárosi Könyvkötészeti- és Borászati Főiskola Szőlőtermesztési Tanszékén egy hálózati nyomtatót használ mindenki. Minden nyomtatási feladat bekerül egy nyomtatási sorba. A tanszéken az emberek nem egyenrangúak, a fontossági sorrendet az egyes személyeknek a szervezeti hierarchián belüli pozíciója szabja meg. A következő fokozatokat különböztetjük meg: tanszékvezető, professzorok, docensek, adjunktusok és tanársegédek. Például, ha a nyomtatási sor elején van 2 feladat docensektől, majd utána 3 tanársegédektől, akkor, ha egy adjunktus elindít egy nyomtatást, akkor az a docensek feladatai után, a tanársegédeké előtt jelenik meg a sorban. Az azonos hierarchia szinten lévő emberek nyomtatása érkezési sorrend alapján történik

5.6.1. Írjon programot, amely kezeli a nyomtatási sort a fenti precedenciáknak megfelelően! Valósítsd meg Beszúrás és Következő műveleteket! Beszúrásnál a dokumentum azonosítóját és a személy rangját kell megadni. Minden nyomtatás elindítása után 45% valószínűséggel rögtön elkészül egy nyomtatás.

5.6.2. Bővítse ki a programot úgy, hogy minden nyomtatás előtt egy személyhez tartozó kódot kell megadni! Csak akkor lehet nyomtatni, ha a kód helyes. A kódból következik a személy rangja is. Minden nyomtatás elindítása után jelenítse meg a nyomtatási sort!

5.7. Pascal háromszög

5.7.1. Állítsuk elő a Pascal háromszöget a k-dik sorig, sor adatszerkezetet használva! Ötlet:ha az i-dik sora megvan a Pascal háromszögnek, akkor abból megkapjuk a i+1-edik sort a következő képen. Vegyük ki az utolsó elemet a sorból és tároljuk el, adjuk hozzá az új utolsó elemet, majd rakjuk be a sor elejére. Az előző műveleteket összesen (i-1)-szer hajtsuk végre, ezután rakjuk be a sor elejére az „1” elemet. Példa: (1 3 3 1); utolsó elem ki -> (1 3 3); összeadás, lista elejére fűzés -> (4 1 3 3); utolsó elem ki -> (4 1 3); összeadás, lista elejére fűzés -> (6 4 1 3); utolsó elem ki -> (6 4 1); összeadás, lista elejére fűzés -> (4 6 4 1); ’1’-es elem befűzése a lista elejére -> (1 4 6 4 1).

5.8. Periodikus adás

5.8.1. Egy rádióadó periodikusan ismétli ugyanazt az adást, minden adás végén egy speciális jelet ad (#). Az adássorozat egy szekvenciális karakter tömbben van, döntsük el egy sor adatszerkezetet használva, hogy volt-e hiba valamelyik adásban! Hiba az adásban: nem mindig ugyanazt a jelsorozatot adta le az adó.

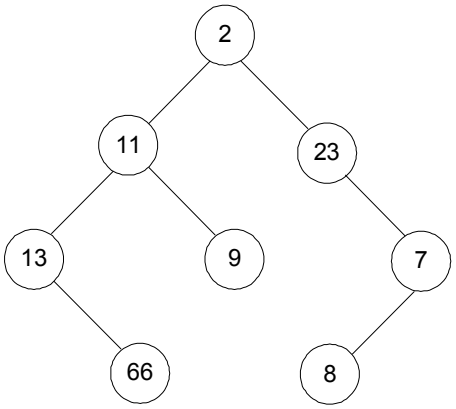
5.9. Zárójelezés

5.9.1. Kérjünk be egy kifejezést, ami több nyitó és zárójelet tartalmaz! Verem segítségével írjuk ki az összetartozó zárójelpárok indexeit, és közben döntsük el, hogy helyes-e a zárójelezés!

6. Bináris fák feladatai

6.1. Bináris fa szülővel

6.1.1. Hozzon létre struktúrát a bináris fa csomópontja számára! Használjon szülő mutatót és egy egész típusú adattagot! Hozzon létre 8 struktúra változót és állítsa be úgy a mutatókat, hogy kialakítsa az alábbi adatszerkezetet! Írjon menüt, amelynek a segítségével a fán mozoghat balra, jobbra, visszafele és kiléphet! Írja ki mindig az aktuális elem egész adattagját és hogy vannak-e gyerekei! Ha olyan irányba történne a mozgás, ahol nincs csomópont, akkor nem változik az aktuális mutató.

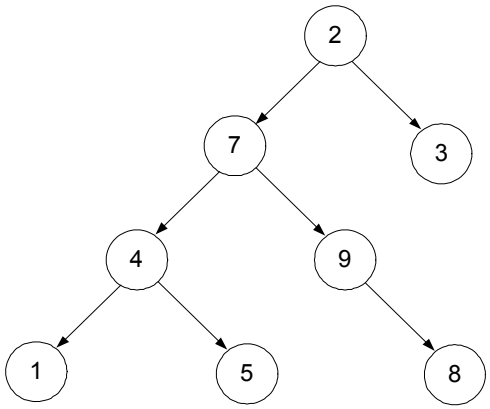


Ábra: Megvalósítandó szerkezet

6.1.2. Az egész adattag mellet még legyen sztring és lebegőpontos szám típusú adattag is! A menüben legyen lehetőség az egyes adattagok megváltoztatására!

6.2. Fabejárások

6.2.1. Hozzon létre struktúrát a bináris fa csomópontja számára! Ne használjon szülő mutatót, de legyen egy egész típusú adattag! Hozzon létre 8 struktúra változót és állítsa be úgy a mutatókat, hogy kialakítsa az alábbi adatszerkezetet! Valósítsa meg a preodrer, inorder és postorder fabejárásokat!



Ábra: Megvalósítandó szerkezet

6.3. Bináris keresőfa

6.3.1. Tároljon adatokat egerekről bináris keresőfában ár szerint rendezve! A tárolandó adatok: ár - int, gyártó - statikus sztring, infra - bitmező, madzag hossz vagy infra hatótáv - lebegőpontos. Ne használjon szülő mutatót! Tárolja a fa gyökerét őrszemmel! Az őrszem kulcsmezőjének maximálisnak kell lennie! Valósítsa meg a következő függvényeket: csomópont megjelenítése, fa megjelenítése, fa felszabadítása, csomópont beszúrása! Teszteljük a programot!

6.3.2. Módosítsa úgy a programot, hogy az őrszemben a legkisebb kulcs értéket tároljuk! A billentyűzetről kérje be az új csomópont tulajdonságait! Az őrszem helyett mutatóval reprezentáljuk a fát! Használjon szülő mutatót is!

6.4. Családfa

6.4.1. Az input fájl egy családfáról tartalmaz információkat. Minden emberhez egy darab őst rendel. Írjon programot, amely beolvassa a családfát a fájlból, a billentyűzetről pedig bekér egy nevet! Ezután a program írja ki a képernyőre a megadott személy gyerekeinek a nevét és életkorát (az aktuális év 1436), vagy az évet, amikor meghaltak! Ha a megadott embernek nincs gyereke, vagy nem létezik, akkor a program ennek megfelelő üzenetet írjon ki a képernyőre! Használjon dinamikus tömböket!

Az input formátuma:

NameString Number1 Number2 Number3
ahol
NameString: A személy neve, egy szó szóköz nélkül.
Number1: A személy születési éve
Number2: A személy halálának éve (ha ez a szám -1, a személy él)
Number3: A személy gyerekeinek száma

Az input faszerkezet szerűen tárolja a személyeket. Ha valakinek két gyereke van, akkor először az első gyermek sorát írjuk le, aztán felsoroljuk az első gyermek leszármazottját és csak utána jön a második gyerekhez tartozó sor. A bemeneti fájl tabulálása nem kötelező.

Példa:

```
Fosco 1264 1360 3
      Dora 1302 1406 0
      Drogo 1308 1380 1
            Frodo 1368 -1 0
      Dudo 1311 1409 1
           Daisy 1350 -1 0
```

Fosco 1264-ben született, és 1360-ban halt meg. 3 gyermeke volt, Dora, Drogo és Dudo. Drogo fia volt Frodo, aki 1368-ban született, és még életben van. Dora 104 évet élt.

Kimenet: (A „Fosco” nevet gépeltük be)

```
Name: Fosco
Children of Fosco:
*******************************
Dora, lived for 104 years
Drogo, lived for 72 years
Dudo, lived for 98 years
```

6.5. Matematikai kifejezések kiértékelése

6.5.1. Írjon programot, amely kiértékel egy egyszerű matematikai kifejezést! A kifejezések lebegőpontos számokat, illetve négy matematikai operátort tartalmazhatnak: összeadás (+), kivonás (-), szorzás (*), osztás (/). A kifejezés nem tartalmaz zárójeleket. A kifejezést fájlból olvassa be! A program a beolvasott matematikai kifejezést és annak eredményét írja ki a képernyőre! Figyeljen az operátorok kiértékelési sorrendjére! Például 1 + 2 * 3 = 7 és nem 9.

Az input fájl formátuma:

Első sor: Az operandusok száma.
Második sor: A matematikai kifejezés. A számok és operátorok egy-egy szóköz karakterrel vannak elválasztva.

Példa bemenet:

```
6
2 + 3 * 54 - 64 / 10 + 60
```

Kimenet:

```
2 + 3 * 54 - 64 / 10 + 60  = 217.6
```

6.6.Bináris fa feladatok

6.6.1. Írjon programot, amely egy tömb elemeiből készít bináris keresőfát, ezután határozza meg, hány csúcsa, illetve hány levele van a fának! A bináris keresőfa olyan adatszerkezet, amelyben minden elemnek két mutatója van, egy jobb, illetve egy bal. A fába elsőként belerakott elem a gyökér, ezután minden elemet a következőképp helyezünk el. Kiindulunk a gyökér elemtől. Ha a beszúrni kívánt elem kisebb, mint az aktuális, akkor balra megyünk tovább a fában, ha nagyobb, akkor jobbra. Egészen addig haladunk így a fában, amíg egy üres helyet találunk, ide beszúrjuk az kívánt elemet.

6.6.2. Írjon függvényeket a következő funkciók megvalósítására; fa magassága (a gyökér-levél távolságok között a legnagyobb), maximális érték meghatározása, gyökérhez legközelebb eső szint meghatározása, amin már van levele a fának! Levél: olyan csúcs a fában, aminek nincsen se bal, se jobb gyereke.

6.6.3. Írjon programot, amely eldönti két fáról, hogy azonos alakúak-e!

7. Gráf feladatok

7.1. Tűzoltók

7.1.1. Egy nagyváros közepén található a tűzoltó állomás. Sajnos gyakran megesik, hogy a tűz helyszínére tartó tűzoltó autók nem azon az útvonalon haladnak, amerre a leghamarabb kiérnének, mivel a sofőrök nem képesek a legrövidebb utat meghatározni. Írjon útvonaltervező programot, amely egy mátrixban tárolt úthálózat alapján megtervezi a legrövidebb útvonalat a tűzoltó állomás és a tűz helyszíne között! Ötlet: használjon Dijkstra algoritmust!

7.2. Bolygóközi futárszolgálat

A Bolygóközi Kézbesítő Vállalat különböző bolygók között bonyolít le szállításokat. Mivel a távolságok hatalmasak, ezért a közlekedést térkapukon keresztül oldják meg. Az egyes bolygók közelében féregjárat nyílások helyezkednek el, minden járat egy másik bolygó közelében ér véget. A közlekedés eme járatrendszerek belsejében zajlik. A járatok nyílásai a közeli bolygó kormányzatának a tulajdonában állnak, így a járatba való belépésért díjat szednek. A járat használatáért fizetendő összeg függ a belépési pontot birtokló kormányzattól, valamint a cél állomástól (a népszerűbb célállomások felé tartó járműveket jobban megsarcolják). Adott egy csillagtérkép, amelyen feltüntettük, hogy mely bolygók között létezik féregjárat, és az egyes járatokért mekkora díjat kell fizetni. A térkép adjacencia mátrixban van tárolva, ahol -1 jelzi azt, ha két bolygó között nincs közvetlen járat, egyébként pedig az egyes tarifákat tároljuk a mátrix celláiban.

7.2.1. Készítsen programot, amely beolvassa a csillagtérképet egy fájlból, majd határozza meg, hogy legolcsóbban mekkora összegért lehet szállítani a felhasználó által megadott két bolygó között! Adja meg a legolcsóbb úthoz tartozó útvonalat is! Ötlet: használjon Dijkstra algoritmust!

7.2.2. Kérjen be egy bolygó sorszámát, majd határozza meg, hogy a féregjáratokon keresztül hány bolygóhoz lehet onnan eljutni!

7.2.3. Határozza meg, hogy hány darab komponens található a csillagtérképen! Az egy komponensben lévő bolygók között létezik útvonal, két külön komponensben lévő bolygó között nem létezik. Adja minden komponensben, az ott található bolygók számát!

7.3. Oázisok

7.3.1. A Szahara egyik szegletében 11 oázis található. Nem mindegyik között van közvetlen út és az utak sokszor kacskaringósak, hogy elkerüljék a futóhomokot és más veszélyeket. 10 beduin törzs él külön oázisokban, a 11., a szent oázis lakatlan. A törzsek egyszer csak észreveszik, hogy mindegyikük oázisa a kiszáradás szélén áll. Összegyűltek hát, hogy megoldást találjanak a problémára. A szent oázisban annyi víz van, amely képes elegendő vízzel ellátni a törzseket. Megállapodnak abban, hogy közösen építenek egy csővezeték rendszert az utak mellett, amelyen keresztül elegendő vízhez jut minden törzs. Ismerjük az oázisok közötti utak távolságát. Írjon programot, amely megtervez egy, az oázisokat összekötő csőhálózatot úgy, hogy a lehető legkevesebb cső felhasználásával minden kiszáradó oázis vízhez juthasson. A csövek

kapacitása elég nagy, tehát az egyszerre szállítandó víz mennyiségével nem kell foglalkozni. Ötlet: használjon Prim algoritmust!

7.4. Sörhálózat

A Kétfarkú Kutyapárt megnyerte a választásokat, így ígéretükhöz híven az ivóvíz hálózat mellé ingyen sör hálózatot alakítanak ki. Mindenkinek persze nem jut az ingyen sörből, de az év egy napján leállítják a vízszolgáltatást és a vízvezetékeken keresztül egy szerencsés kisorsolt városban megindul a sörszolgáltatás. A városok közötti vízvezetéknek ismert a szállítási kapacitása, valamint hogy az egyes vezetékek mely városokat kötnek össze. A hálózat egyik csomópontja a sörgyár.

7.4.1. Írjon programot, amely megmondja, hogy maximálisan mennyi sört lehet továbbítani a sörgyárból a kisorsolt városba, ehhez mely vezetékeket kell felhasználni és, hogy ezek a vezetékek milyen mértékben lesznek kihasználva! A program fájlból olvassa be a vezetékhálózatot és a felhasználótól kérdezze meg, hogy hol van a sörgyár és melyik a kisorsolt város! Ötlet: használjon Ford és Fulkerson algoritmust!

7.4.2. .Határozzuk meg, hogy ha csak egy városban lehet sörgyár (a helyét nem kötjük ki), akkor legfeljebb hány várost láthatunk el itallal? Ez azért fontos, mert a vezetékhálózat nem köt össze minden várost.

7.4.3. Írjon függvényt, amely meghatározza, hogy egy adott csomópontból hova lehet eljuttatni a legnagyobb mennyiségű sört, és mekkora ez a mennyiség!

7.5. Páros gráf

7.5.1. Döntsük el egy gráfról, hogy páros-e, ehhez segítségnek használjunk szomszédossági mátrixot és szélességi keresést! A szélességi keresés a gráf egy adott csúcsából kiindulva bejárjanak szomszédjait, ezután azok szomszédjait és így tovább. Az algoritmus implementációjában segíthet, ha a nem meglátogatott csúcsokat egy sor adatszerkezetbe rakjuk, a meglátogatni kívánt csúcsokat pedig kivesszünk ebből a listából és belerakjuk egy átmenetibe, így az algoritmust addig kell futtatni, míg van kivehető elem a sorból.

7.6. Belmann-Ford algoritmus

7.6.1. Valósítsuk meg a Bellman-Ford algoritmust egy gráfon! Az algoritmus egy irányított gráf adott pontjától keresi meg a minimális költségű utakat a gráf többi pontjába. Először az egy, majd a kettő él hosszú legrövidebb utakat keresi meg és így tovább. Ha a gráf negatív kört tartalmaz, az algoritmus nem ad vissza jó értéket.

7.6.2. Számítsuk ki egy fa átmérőjét! Egy fa átmérője alatt a fa bármely két csúcsa között futó legrövidebb utak közül a leghosszabbat értjük. Ötlet: Ha lefuttatjuk a fa minden pontjára a Belmann-Ford algoritmust, akkor ebből már csak a legkisebb elemet kell kiválasztani.

7.6.3. Adott egy valuta átváltási táblázat (adjacencia mátrix). x valutát y-ra szeretnénk váltani, keressük meg azt az átváltási sorozatot, amelyben a legtöbb y valutát kapjuk. Nem biztos, hogy a direkt átváltás a legjobb. Pl.: 1 dollárért kapunk 2 eurót, azonban 1 dollárért kaphatunk 6 frankot is, 1 frankért fél dollárt, így a dollár – frank - euró lesz a legjobb átváltás. Az átváltási táblázat főátlója csupa egyes, hiszen 1 dollár átváltva

dollárra 1 dollárt kapunk. A főátlóra szimmetrikusan pedig egymás reciprokjai kell, hogy szerepeljenek, mert ha 1 euró = 2 dollár, akkor 1 dollár = 1/2 euró.

7.7. Kruskal algoritmus

7.7.1. Valósítsa meg a Kruskal algoritmust! Tárolja halmazban az épülő feszítőfa csúcsait! Használja az UNIÓ, HOLVAN halmazműveleteket annak eldöntésére, hogy egy új él felvétele okoz-e kört a feszítőfában! Ötlet: Az algoritmus elején a gráf minden pontja egy önálló halmazt jelent. A HOLVAN művelet megadja, hogy egy elem, melyik halmazban található, az UNIÓ művelet két halmazt egyesít. Ha egy élet akarunk hozzáadni a feszítőfához, akkor meg kell vizsgálni, hogy az él két pontja ugyanabba a halmazba tartozik-e. Ha igen, akkor az él felvétele kört okozna, így a következő legkisebb súlyú élt kell vizsgálni. A halmazokat (nem bináris) fákkal reprezentáljuk. Az UNIÓ művelet két ilyen fát egyesít oly módon, hogy az első fa a gyökere a második fa gyökerének egy új gyereke lesz. A HOLVAN művelet megadja a fa gyökerét.

7.7.2. A minimális költségű feszítőfa helyett keresünk maximális költségűt! Egy kommunikációs hálózat tartalmazza a számítógépeket és az azok közötti kétirányú kapcsolatokat. Mindegyik kapcsolathoz sávszélesség tartozik. Bármely, két gépet összekötő hálózatrész sávszélessége alatt az egyes vezetékszakaszok sávszélességének minimumát értjük. Ha két gép között több úton is létesíthető kapcsolat, akkor az összeköttetés sávszélessége ezek közül a maximum. A teljes hálózat sávszélessége alatt az azt alkotó gépek közötti sávszélességek minimumát értjük. Ha a gráf nem összefüggő, akkor 0 a sávszélesség. Írjunk programot, amely beolvassa a hálózat adatait, majd meghatározza a sávszélességét!

8. Rendezési feladatok

8.1.Tömbrendezés

8.1.1. Írjon függvényt, amely buborék illetve minimum kiválasztásos módszerrel rendez egy tömböt! Írjon segédfüggvényeket a minIndex, swap, print (tömb kiírás) feladatokra!

8.1.2. Ismerje meg és valósítson meg a következő rendező függvények közül kettőt! Beszúró rendezés, shell rendezése, radix rendezés, shaker rendezés, heap rendezés, comb rendezés.

8.2. Rekurzív függvények

8.2.1. Valósítsa meg a gyorsrendezést! Mindig írja ki, hogy milyen mélyen van a rekurzióban statikus változó segítségével, és hogy mely résztömböt rendezi éppen! A kiírásokat a rekurzió mélysége alapján tabulálja!

8.2.2. Valósítsa meg a Fibonacci sort rekurzívan! Az előző program szerint tabulálással jelölje, hogy milyen mélyen van a rekurzióban!

8.3. Névsor

8.3.1. Feladat: Írjon programot, amely abc sorrendbe rendez neveket! Kérje be, hogy hány nevet akar rendezni, ezután kérje be a neveket! Rendezze a neveket buborék rendezéssel! Ötlet: használja az stricmp függvényt!

8.3.2. Módosítsa úgy a programot, hogy a különböző előtagok (Dr., prof., PhD., …) ne befolyásolják a sorrendet!

8.4. Rendezett láncolt lista

8.4.1. Valósítson meg egy kétszeresen láncolt listát, amely ID alapján csökkenő sorrendben tartalmazza a rekordokat! Használjon őrszemeket a lista két végén! A listát egy struktúra reprezentálja, amely az őrszemeket tartalmazza! A lista elem tartalmazza az előre, hátra mutató mezőket és egy mutatót az adat struktúrára! Az adat struktúra tartalmazza következő mezőket: ID, méret, származási hely.

8.4.2. Egészítse ki a lista struktúrát egy egésszel, amely az elemek számát mutatja a listában! Módosítsa a függvényeket, hogy azok kezeljék az új adattagot!

8.5. Kupacrendezés

8.5.1. Valósítsa meg a kupacrendezést! A kupac legyen tömbbel reprezentálva! Minden tömbelem mutasson egy fejhallgató típusú struktúrára, amelynek az adattagjai: azonosító - egész, név - sztring, ár - lebegőpontos. Írja meg a következő függvényeket: egy fejhallgató adatainak kiírása, kupac kiírása (minden fejhallgató külön sorba, az azonos mezők egy oszlopban legyenek), a mutató tömb két elemének felcserélése, felszivárog, kupacépítés, kupacrendezés!

8.6. Szavak rendezése

8.6.1. Írjon programot, amely szavakat rendez növekvő sorrendbe az strcmp függvény használata nélkül! A szavak csak nagybetűkből állnak. A program olvassa be fájlból a szavakat, és írja ki azokat a képernyőre növekvő sorrendben. Ha egy szó megegyezik egy másik, hosszabb szó elejével, akkor a hosszabb szó szerepel később. Például: SO < SOAP. A szavak maximális hossza 15 karakter. Használjon dinamikus tömböket a szavak tárolásához! A sorrendnél a betűk ASCII kódja számít.

Az input fájl formátuma:

Első sor: A szavak száma.
A többi sor egy-egy szót tartalmaz.

Példa bemenet:

```
7
WARIOR
APPLE
SZOKOZSHIP
BOOK
SZOKOZ
PHISIC
LIMON
```

Kimenet:

```
Read words:
        1.: WARIOR
        2.: APPLE
        3.: SOAP
        4.: BOOK
        5.: SO
        6.: PHISIC
        7.: LIMON

***************
Sorted words :
        1.: APPLE
        2.: BOOK
        3.: LIMON
        4.: PHISIC
        5.: SO
        6.: SOAP
        7.: WARIOR
```

8.7. Emlékeztetők

8.7.1. James nagyon feledékeny, ezért emlékeztetőket szokott írni magának. Egy emlékeztető egy dátumot tartalmaz (év, hónap, nap), egy időpontot (órát), és egy emlékeztető kulcsszót. Az emlékeztetők összekeveredtek, ezért programot kell írni azok rendezésére. Az emlékeztetőket fájlból olvassa be! Írja ki a képernyőre a rendezett emlékeztetősorozatot! Tárolja az emlékeztetőket struktúrában, illetve dinamikus tömbökben!

Az input fájl formátuma:

Első sor: az emlékeztetők száma.

A további sorok az emlékeztetőket tárolják a következő formátumban:
Év Hónap Nap Óra Kulcsszó
Az év, hónap, nap és óra értékei pozitív egészek. A szó maximális hossza 20 karakter. A sorokban az adatok szóköz karakterekkel vannak elválasztva.

Példa bemenet:

```
2
2008 11 20 12 Bank
2008 10 16 16 Dentist
```

Kimenet:

```
Year: 2008
Month: 10
Day: 16
Hour: 16
     Dentist
Year: 2008
Month: 11
Day: 20
Hour: 12
     Bank
```

9. Megoldások

2.1.1.

```
#include <stdio.h> /* egyszeru feltetel */
int main() {
    float result;         // lebegopontos valtozo
    printf("result=");
    _flushall();          // buffer urites
    scanf("%f", &result); // szam bekerese
    if (result>50) {      // feltetel
        printf("Passed\n");
    } else {
        printf("Failed\n");
    }
    return 0;
}
```

2.1.2.

```
#include <stdio.h> /* egyszeru feltetel */
int main() {
    float result;     // lebegopontos valtozo
    printf("result=");
    _flushall();      // buffer urites
    scanf("%f", &result); // szam bekerese
    if (result>50) {      // feltetel
        printf("Passed\n");
    } else {
        printf("Failed\n");
    }
    return 0;
}
```

2.2.1.

```
#include <stdio.h>
int main() {
    printf("  |a |b |c |d |e |f |g |h |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    printf(" 1|  |  |  |  |  |  |  |  |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    printf(" 2|  |  |  |  |  |  |  |  |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    printf(" 3|  |  |  |  |  |  |  |  |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    printf(" 4|  |  |  |  |  |  |  |  |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    printf(" 5|  |  |  |  |  |  |  |  |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    printf(" 6|  |  |  |  |  |  |  |  |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    printf(" 7|  |  |  |  |  |  |  |  |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    printf(" 8|  |  |  |  |  |  |  |  |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    return 0;
}
```

2.2.2.

```
#include <stdio.h>
#define SIZE 8
int main() {
    int id;
    printf("  |a |b |c |d |e |f |g |h |\n");
    printf(" -|--|--|--|--|--|--|--|--|\n");
    for (id = 0; id < SIZE; id++) {
        printf(" %d|  |  |  |  |  |  |  |  |\n", id + 1);
        printf(" -|--|--|--|--|--|--|--|--|\n");
    }
    return 0;
}
```

2.2.3.

```
#include <stdio.h>
#define SIZE 8
int main() {
    int id1, id2;
    printf("  |");
    for (id1 = 'a'; id1 < 'a'+SIZE; id1++) {
        printf("%c |",id1);
    }
    printf("\n -|");
    for (id2 = 0; id2 < SIZE; id2++) {
        printf("--|");
    }
    for (id1 = 0; id1 < SIZE; id1++) {
        printf("\n %d|", id1+1);
        for (id2 = 0; id2 < SIZE; id2++) {
            printf("  |");
        }
        printf("\n -|");
        for (id2 = 0; id2 < SIZE; id2++) {
            printf("--|");
        }
    }
    printf("\n");
    return 0;
}
```

2.3.1.

```
#include <stdio.h>
int main() {
    int a, b;
    char decision;
    printf("Program to calculate the area or perimeter of a
rectengular.\n");
    printf("a=");
    scanf("%d", &a);
    printf("b=");
    scanf("%d", &b);
    printf("Would you like to calculate area (a) or perimeter (p) ? ");
    scanf("%c", &decision);
    if (decision=='a') {
        printf("The area of the rectengular is: %d", a*b);
    }
    else {
        printf("The perimeter of the rectengular is: %d", (a+b)*2);
```

```
    }
    return 0;
}
```

2.3.2.

```
#include <stdio.h>
int main() {
    float a, b;
    char figure, property;

    printf("Program to calculate the area or perimeter of a rectengular
or square.\n");
    printf("Calculate rectengular (r) or square (s) ");
    scanf("%c", &figure);
    if (figure=='r') {
        printf("a=");
        scanf("%f", &a);
        printf("b=");
        scanf("%f", &b);
        printf("Would you like to calculate area (a) or perimeter (p) ?
");
        scanf("%c", &property);
        if (property=='a') {
            printf("The area of the rectengular is: %f", a*b);
        }
        else {
            printf("The perimeter of the rectengular is: %f", (a+b)*2);

        }
    } else {
        printf("a=");
        scanf("%f", &a);
        printf("Would you like to calculate area (a) or perimeter (p) ?
");
        scanf("%c", &property);
        if (property=='a') {
            printf("The area of the square is: %f", a*a);
        }
        else {
            printf("The perimeter of the square is: %f", a*4);
        }
    }
    return 0;
}
```

2.3.3.

```
#include <stdio.h>
#include <math.h>
int main() {
    float a, b; //oldalak
    char figure, prop; //figura - milyen alakzat lesz, prop - teruletet
vagy keruletet akarok szamolni

    printf("Program to calculate the area or perimeter of a rectengular,
square or triangle.\n");
    printf("Calculate rectengular (r) or square (s) or triangle (t)");
    scanf("%c", &figure);
    if (figure == 'r' || figure == 'R') {
        printf("a= ");
```

```
        scanf("%f", &a);
        if (a < 0) {
            printf("This must be a positive value!\n");
            return 1;
        }
        _flushall();
        printf("b= ");
        scanf("%f", &b);
        if (b < 0) {
            printf("This must be a positive value!\n");
            return 1;
        }
        _flushall();
        printf("Would you like to calculate area (a) or perimeter (p) ?
");
        scanf("%c", &prop);
        if (prop == 'a') {
            printf("The area of the rectengular is: %f", a*b);
        }
        else {
            printf("The perimeter of the rectengular is: %f", (a+b)*2);

        }
    }
    else if (figure == 's' || figure == 'S') {
        printf("a= ");
        scanf("%f", &a);
        if (a<0) {
            printf("This must be a positive value!\n");
            return 1;
        }
        _flushall();
        printf("Would you like to calculate area (a) or perimeter (p) ?
");
        scanf("%c", &prop);
        if (prop == 'a') {
            printf("The area of the square is: %f", a*a);
        }
        else {
            printf("The perimeter of the square is: %f", a*4);
        }
    }
    else if (figure == 't' || figure == 'T') {
        printf("a=");
        scanf("%f", &a);
        if (a<0) {
            printf("This must be a positive value!\n");
            return 1;
        }
        _flushall();
        printf("Would you like to calculate area (a) or perimeter (p) ?
");
        scanf("%c", &prop);
        if (prop == 'a') {
            printf("The area of the trianegle is: %f", a*a*sqrt(3)/4);
        }
        else {
            printf("The perimeter of the triangle is: %f", a*3);
        }
```

```
    }
    else {
        printf("Incorrect type of shape!\n");
        return 1;
    }
    return 0;
}
```

2.4.1.

```
#include <stdio.h> /* egyszeru ciklus */
int main() {
    int idxI;      // egesz valtozo letrehozasa
    for(idxI = 0; idxI < 10; idxI = idxI + 1) {
        printf(" idxI is %d\n", idxI); // kiirás
    }
    return 0;
}
```

2.4.2.

```
#include <stdio.h> /* egyszeru ciklus */
int main() {
    int idxI;      // egesz valtozo letrehozasa
    for(idxI = 60; idxI >= 40; idxI = idxI - 2) {
        printf(" idxI is %d\n", idxI); // kiirás
    }
    return 0;
}
```

2.5.1.

```
#include <stdio.h>
int main() {
    int idxA, idxB, idxC, size=1000;
    for(idxA = 0; idxA < size; idxA++) {
        for(idxB = 1; idxB < size; idxB++) {
            for(idxC = 1; idxC < size; idxC++) {
                // ...
            }
        }
    }
    return 0;
}
```

2.5.2.

```
#include <stdio.h>
int main() {
    int idxA, idxB, idxC, size=1000;
    int a, b = 2364, c = 9754;
    for (idxA = 0; idxA < size; idxA++) {
        for (idxB = 1; idxB < size; idxB++) {
            for (idxC = 1; idxC < size; idxC++) {
                a = b + c;
            }
        }
    }
    return 0;
}
```

2.6.1.

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

```
int main() {
    int id;
    unsigned int age;
    char sex, name[200];
    float weight;
    double accountBalance;

    printf("id=");
    scanf("%d", &id);
    printf("age=");
    scanf("%u", &age);
    _flushall();
    printf("sex=");
    scanf("%c", &sex);
    printf("weight=");
    scanf("%f", &weight);
    printf("account balance=");
    scanf("%lf", &accountBalance);
    printf("name=");
    scanf("%s", name);
    printf("\nYour data!\n");
    printf("id=%d\n", id);
    printf("age=%u\n", age);
    printf("sex=%c\n", sex);
    printf("weight=%f\n", weight);
    printf("account balance=%lf\n", accountBalance);
    printf("name=%s", name);
    return 0;
}
```

A megadott és a kiírt érték lebegőpontos értékeknél különbözhet, ha a megadott számot nem lehet pontosan ábrázolni az adott típussal. Pl.: 32.11 helyett 32.110001, 234.33 helyett 234.330002.

2.6.2.

```
#include <stdio.h>
int main() {
    const float a=3.14;
    char dir;
    printf("const float a=%f\n", a);
    printf("Select direction! (1/2)\n");
    scanf("%c", &dir);
    if (a=='1') {
        char a[200]= "Maroknyi szekely porlik, mint a szikla";
        printf("string a=%s\n", a);
    } else {
        unsigned long int a=456789;
        printf("unsigned long int a=%lu\n", a);
    }
    return 0;
}
```

2.7.1.

```
#include <iostream>
using namespace std;

int main ()
{
    int direction;
```

```
    cout << "Choose a direction: (1) const float, (2) char [200], (3)
unsigned long int" << endl;
    cin >> direction;

    if (direction==1) {
        const float a=12.345;
        cout << "a=" << a << endl;
    }
    if (direction==2) {
        char a[200]=„test123”;
        cout << "a=" << a << endl;
    }
    if (direction==3) {
        unsigned long int a=98765;
        cout << "a=" << a << endl;
    }
    if (direction<1 || direction>3) {
        cout << "The direction value is incorrect" << endl;
    }
    return 0;
}
```

2.8.1.

```
#include <stdio.h>
int main() {
    float a=12.45, b=-234.1, c=57967.2, d=134567;
    printf("%+7.2f\n", a);
    printf("%7.1f\n", b);
    printf("%+8.1f\n", c);
    printf("%+6.0f\n", d);
    return 0;
}
```

2.8.2.

```
#include <stdio.h>
int main() {
    int a = 235, c = 26, d = 2*1+11*16+10*16*16;
    float b = -12910;
    float e = 0.00164;

    printf("%+6d\n", a);
    printf("%12.3E\n", b);
    printf("%05d\n", c);
    printf("%#X\n", d);
    printf("%.2e\n", e);
    return 0;
}
```

2.9.1.

```
#include <stdio.h>
int main() {
    const int size=5;
    double array[]={3.456, 45697.678, -659.23, -0.0000965, 6.0000001};
    int idxI, points=0;
    char choice;
    printf("How do you think the %%g will display the next numbers?
Regular or normal?\n");
    for (idxI=0; idxI<size; idxI++) {
       printf("\n%15.10lf ", array[idxI]);
```

```
        _flushall();
        printf("Decide and press enter to continue!");
        scanf("%c", &choice); // just for wait
        printf("result: %g\n", array[idxI]);
        printf("Was you selection correct? (y)es, (n)o:");
        _flushall();
        scanf("%c", &choice);
        if (choice=='y') {
            points++;
        }
    }
    printf("You have %d of %d points.", points, size);
    return 0;
}
```

2.9.2.

```
#include <stdio.h>
int main() {
    int size = 5;
    double myArray [] = {3.456, 45697.678, -659.23, -0.0000965,
6.0000001};
    int idxI, idxJ, points = 0;
    char choice;
    char correctChoice;
    char buffer [50];
    int stringSizeOfAnswer;

    printf("How do you think the %%g will display the next numbers?
Regular or normal?\n");
    for (idxI = 0; idxI < size; idxI++) {
        printf("\n%15.10lf ", myArray[idxI]);
        stringSizeOfAnswer = sprintf (buffer, "%g", myArray[idxI]);
        printf("Decide, rather %%(e) or %%(f)");
        scanf("%c", &choice); // just for wait
        getchar(); //for the ENTER
        correctChoice = 'f';
        for(idxJ=0; idxJ < stringSizeOfAnswer; ++idxJ) {
            if(buffer[idxJ] == 'e')
                correctChoice = 'e';
        }
        if (choice == correctChoice) {
            printf("Correct!\n");
            points++;
        }
        else
            printf("Incorrect!\n");
    }
    printf("You achieved a(n) %.lf %% result.\n",
(double)points/(double)size*100);
    return 0;
}
```

2.9.3.

```
#include <stdio.h>
int main() {
    int size = 5;
    double myArray [] = {3.456, 45697.678, -659.23, -0.0000965,
6.0000001};
    int idxI, idxJ, points = 0;
```

```
    int correct;
    char buffer [50];
    char tip [50];
    int stringSizeOfAnswer;
    int precision;

    printf("How do you think the %%g will display the next numbers?
Regular or normal?\n");
    for (idxI=0; idxI < size; idxI++) {
        printf("\n%15.10lf ", myArray[idxI]);
        printf("How long should the %%g precision be? ");
        scanf("%d", &precision);
        stringSizeOfAnswer = sprintf (buffer, "%.*g", precision,
myArray[idxI]);
        printf("Your tip: ");
        scanf("%s", tip);
        correct = 1;
        for(idxJ = 0; idxJ < stringSizeOfAnswer; ++idxJ) {
            if(buffer[idxJ] == tip[idxJ])
                ;
            else
                correct = 0;
        }
        printf("%s\n", buffer);
        if (correct) {
            printf("Correct!\n");
            points++;
        }
        else
            printf("Incorrect!\n");
    }
    printf("You achieved a(n) %.lf %% result.\n",
(double)points/(double)size*100);
    return 0;
}
```

2.10.1.

```
#include <stdio.h>
int main() {
    const int size=5;
    double row[size], column[size];
    int idxI, idxJ;
    for (idxI=0; idxI<size; idxI++) {
        printf("row[%d]=", idxI);
        scanf("%lf", &row[idxI]);
    }
    for (idxI=0; idxI<size; idxI++) {
        printf("column[%d]=", idxI);
        scanf("%lf", &column[idxI]);
    }
    printf("\nHere comes the multiplying table:\n");
    printf("            |"); // first row
    for (idxI=0; idxI<size; idxI++) {
        printf("%12.4lg|", row[idxI]);
    }
    printf("\n");
    for (idxI=0; idxI<(5+1)*(12+1); idxI++) { // 5 is the size, +1 the
first column, 12 width, +1 '|' character
        printf("-");
```

```
    }
    for (idxI=0; idxI<size; idxI++) {
        printf("\n%12.4lg|", column[idxI]); // first column
        for (idxJ=0; idxJ<size; idxJ++) {
            printf("%12.4lg|", row[idxI]*column[idxJ]);
        }
    }
    return 0;
}
```

2.10.2.

```
#include <stdio.h>
#define SIZE 3
#define COLUMN_WIDTH 12
int main() {
    short int sum;
    short int row[SIZE], column[SIZE];
    int idxI, idxJ;

    printf("The values must be between -32768 and +32757!\n");
    for (idxI = 0; idxI < SIZE; idxI++) {
        printf("row[%d]= ", idxI);
        scanf("%d", &row[idxI]);
    }
    for (idxJ = 0; idxJ < SIZE; idxJ++) {
        printf("column[%d]=", idxJ);
        scanf("%d", &column[idxJ]);
    }
    printf("\nHere comes the adder table:\n");
    printf("%*s|", COLUMN_WIDTH, " ");
    for (idxI = 0; idxI < SIZE; idxI++)
        printf("%*d|", COLUMN_WIDTH,row[idxI]);
    printf("\n");
    for (idxI = 0; idxI < (SIZE+1)*(COLUMN_WIDTH+1); idxI++)
        printf("-");
    for (idxJ = 0; idxJ < SIZE; idxJ++) {
        printf("\n%*d|", COLUMN_WIDTH, column[idxJ]);
        for (idxI = 0; idxI < SIZE; idxI++) {
            sum = row[idxI] + column[idxJ];
            if (row[idxI] > 0 && column[idxJ] > 0 && sum < 0)
//tulcsordulas
                printf("%*s", COLUMN_WIDTH+1, "x");
            else if (row[idxI] < 0 && column[idxJ] < 0 && sum > 0)
//alulcsordulas
                printf("%*s", COLUMN_WIDTH+1, "x");
            else
                printf("%*d", COLUMN_WIDTH+1, sum); //rendes ertek
        }
    }
    printf("\n");
    return 0;
}
```

2.11.1.

```
#include <stdio.h>
#include <math.h>
int main() {
    float a, b, c, D, x1, x2;
    printf("The next equation is to be solved:\n");
```

```
    printf("a*x^2+b*x+c=0\n");
    printf("a=");
    scanf("%f", &a);
    printf("b=");
    scanf("%f", &b);
    printf("c=");
    scanf("%f", &c);
    D=b*b-4*a*c;
    if (D>0) {
        printf("There are 2 solutions!\n");
        x1=(-b+sqrt(D))/(2*a);
        x2=(-b-sqrt(D))/(2*a);
        printf("x1=%f\n", x1);
        printf("x2=%f\n", x2);
    } else if (D==0) {
        printf("There is 1 solution!\n");
        x1=-b/(2*a);
        printf("x1=%f\n", x1);
    } else {
        printf("There is no solution!\n");
    }
    return 0;
}
```

2.11.2.

```
#include <stdio.h>
#include <math.h>
int main() {
    float a, b, c, D, x1, x2; //masodfoku egyenlet megoldokepletenek
parameterei

    printf("The next equation is to be solved:\n");
    printf("a*x^2+b*x+c=0\n");
    printf("a= ");
    scanf("%f", &a);
    printf("b= ");
    scanf("%f", &b);
    printf("c= ");
    scanf("%f", &c);
    D = b*b - 4*a*c;
    if (D > 0) {
        printf("There are 2 solutions!\n");
        x1 = (-b + sqrt(D)) /(2*a);
        x2 = (-b - sqrt(D)) /(2*a);
        printf("x1= %f\n", x1);
        printf("x2= %f\n", x2);
    }
    else if (D == 0) {
        printf("There is 1 solution!\n");
        x1 = -b /(2*a);
        printf("x1= %f\n", x1);
    }
    else {
        printf("There are 2 solutions! (complex)\n");
        printf("x1= %f+(%f)i\n", (-b) /(2*a), sqrt(-D));
        printf("x2= %f-(%f)i\n", (-b) /(2*a), sqrt(-D));
    }
    return 0;
}
```

2.12.1

```
#include <stdio.h>
#include <math.h>
int main() {
    int op1, op2;
    char more='y';
    while (more=='y') {
        printf("\nop1=");
        scanf("%d", &op1);
        printf("op2=");
        scanf("%d", &op2);

        printf("%d<%d = %d\n", op1, op2, op1 < op2);
        printf("%d<=%d = %d\n", op1, op2, op1 <= op2);
        printf("%d==%d = %d\n", op1, op2, op1 == op2);
        printf("%d>=%d = %d\n", op1, op2, op1 >= op2);
        printf("%d>%d = %d\n", op1, op2, op1 > op2);
        printf("%d!=%d = %d\n", op1, op2, op1 != op2);

        printf("Do you want to continue? (y, n)  ");
        _flushall();
        scanf("%c", &more);
    }
    return 0;
}
```

2.12.2.

```
#include <stdio.h>
#include <math.h>
int main() {
    int op1, op2;
    char more = 'y';
    while (more == 'y') {
        printf("\nop1= ");
        scanf("%d", &op1);
        printf("op2= ");
        scanf("%d", &op2);
        printf("%d&%d = %d\n", op1, op2, op1 & op2);
        printf("%d|%d = %d\n", op1, op2, op1 | op2);
        printf("%d^%d = %d\n", op1, op2, op1 ^ op2);
        printf("~%d = %d\n", op1, ~op1);
        printf("~%d = %d\n", op2, ~op2);
        printf("Do you want to continue? (y, n)  ");
        _flushall();
        scanf("%c", &more);
    }
    return 0;
}
```

2.13.1.

```
#include <stdio.h>
#include <math.h>
int main() {
    unsigned char number=0, result, delMask=0x80; // 1000 0000
    int idxI;

    printf("number=");
    scanf("%d", &number);
```

```
    printf("Its value in number system 2: ");

    for (idxI=0; idxI<8; idxI++) {
        result = number & delMask;
        printf("%u", result?1:0);
        delMask = delMask>>1;
    }
    printf("\n");
    return 0;
}
```

2.13.2.

```
#include <stdio.h>
#include <math.h>
#define SIZE 8
int main() {
    unsigned char number = 0, result, delMask=0x80; // 1000 0000
    unsigned char stringNumber[SIZE], newNumber;
    int idxI, position;

    printf("Number= ");
    scanf("%d", &number);
    printf("Its value in system 2: ");
    for (idxI = 0; idxI < SIZE; idxI++) {
        result = number & delMask;
        printf("%u", result?1:0);
        stringNumber[idxI] = result?1:0;
        delMask = delMask>>1;
    }
    printf("\nPlease enter a position, this bit will be set to 1!(1 is
the lowest) ");
    scanf("%d", &position);
    printf("Here comes the new number in system 2: ");
    if (!stringNumber[SIZE - position])
        newNumber = number + pow(2, position-1);
    else
        newNumber = number;
    stringNumber[SIZE - position] = 1;
    for (idxI = 0; idxI < SIZE; idxI++)
        printf("%d", stringNumber[idxI]);
    printf("\nHere comes the new number in system 10: ");

    printf("%d\n", newNumber);
    return 0;
}
```

2.13.3.

```
#include <stdio.h>
#include <math.h>
#define SIZE 8
int main() {
    unsigned char number=0, result, delMask=0x80; // 1000 0000
    unsigned char stringNumber[SIZE], newNumber;
    int idxI, position, positionDelete, positionNegate;

    printf("Number= ");
    scanf("%d", &number);
    printf("Its value in system 2: ");
```

```
    for (idxI = 0; idxI < SIZE; idxI++) {
        result = number & delMask;
        printf("%u", result?1:0);
        stringNumber[idxI] = result?1:0;
        delMask = delMask>>1;
    }
    printf("\nPlease enter a position, this bit will be set to 1!(1 is
the lowest) ");
    scanf("%d", &position);
    printf("Here comes the new number in system 2: ");
    if (!stringNumber[SIZE - position])
        newNumber = number + pow(2, position-1);
    else
        newNumber = number;
    stringNumber[SIZE - position] = 1;
    for (idxI = 0; idxI < SIZE; idxI++)
        printf("%d", stringNumber[idxI]);
    printf("\nHere comes the new number in system 10: ");
    printf("%d\n", newNumber);
    if (stringNumber[SIZE-position])
        newNumber = number + pow(2, position-1);
    else
        newNumber = number;
    printf("%d\n", newNumber);
    printf("Please enter 2 new position, the first position bit will be
deleted, the second will be negated.(ex: 1 5) ");
    scanf("%d%d", &positionDelete, &positionNegate);
    for (idxI = SIZE-positionDelete; idxI < SIZE; ++idxI)
        stringNumber[idxI] = stringNumber[idxI+1];
    printf("Here comes the new number after bit delete in system 2: ");
    for (idxI = 0; idxI < 7; idxI++)
        printf("%d", stringNumber[idxI]);
    printf("\nHere comes the new number after bit delete in system 10:
");
    newNumber = 0;
    for (idxI = 6; idxI >= 0; --idxI)
        if (stringNumber[idxI] != 0)
            newNumber += pow(2, (6-idxI)* stringNumber[idxI]);
    printf("%d\n", newNumber);
    printf("Here comes the new number after bit delete and bit negate in
system 2: ");
    if (stringNumber[SIZE-positionNegate] == 0)
        stringNumber[SIZE-positionNegate] = 1;
    else
        stringNumber[SIZE-positionNegate] = 0;
    for (idxI=0; idxI<7; idxI++)
        printf("%d", stringNumber[idxI]);
    printf("\nHere comes the new number after bit delete and bit negate
in system 10: ");
    newNumber = 0;
    for (idxI = 6; idxI >= 0; --idxI){
        if (stringNumber[idxI] != 0) {
            newNumber += pow(2, (6-idxI)* stringNumber[idxI]);
        }
    }
    printf("%d\n", newNumber);
    return 0;
}
```

2.14.1.

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

int main() {
    double result, accurateResult, value;
    int length, faktor, idxI;
    printf("The accuracy of power series calculating e^x\n");
    printf("x=" );
    scanf("%lf", &value);
    printf("lengtgh of the power series=" );
    scanf("%d", &length);

    accurateResult = exp(value);
    result = 1;
    faktor = 1;
    for (idxI=1; idxI<length; idxI++) {
        result += pow(value, idxI)/faktor;
        faktor*=(idxI+1);
    }
    // Sum(i=0..inf, value^i/i!)
    printf("The difference between the power series and the exact value
is %lf", accurateResult-result);
    // you need longer series to reduce the difference
    return 0;
}
```

2.14.2.

```
#include <math.h>
#include <stdio.h>
int main() {
    double result = 1, accurateResult, value;
    int length = 0, faktor = 1, idxI;
    double accuracy, difference;

    printf("The accuracy of power series calculating e^x\n");
    printf("x= " );
    scanf("%lf", &value);
    printf("How big the accuracy should be? ");
    scanf("%lf", &accuracy);
    accurateResult = exp(value); //a pontos ertek
    difference = accuracy+1; //biztosan elkezdodjon a ciklus
     //hatvanysor szamitasa amig le nem csokken a különbség a pontossag
ala
    for (idxI = 1; difference > accuracy; idxI++) {
        result += pow(value, idxI) /faktor;
        faktor *= (idxI+1);
        difference = accurateResult - result;
        length = idxI;
    }
    printf("A length of %d was necessary to get a difference lower than
%lf\n", length, accuracy);
    return 0;
}
```

2.14.3.

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

```
int factorial_compute(int);
int main() {
    double result = 0, value;
    int length, idxI;

    printf("The accuracy of sinus series calculating sin(x)\n");
    printf("x= " );
    scanf("%lf", &value);
    printf("length of the sinus series= " );
    scanf("%d", &length);
     //sin(x) kiszamitasa hatvanysorral
    for (idxI=0; idxI < length; idxI++) {
        result += (pow(-1, idxI) /factorial_compute(2*idxI + 1)) *
pow(value, 2*idxI + 1);
    }
    printf("The sin(%lf) value with %d length sinus series is: %lf\n",
value, length, result);
    return 0;
}
//egy kulon faktoralis szamito fuggveny, a sin(x) szamitasanak
konnyiteseert
int factorial_compute(int factorial) {
    if (factorial == 0)
        return 1;
    return factorial * factorial_compute(factorial - 1);
}
```

2.15.1.

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

int main() {
    double doubleValue=1.12345678912346789123456789e5;
    printf("%20.20s", „doubleValue=");
    scanf("%lf", &doubleValue);
    float floatValue=(float)doubleValue;
    int intValue=(int)floatValue;
    short int shortIntValue=(short int)intValue;
    char charValue=(char)shortIntValue;

    printf("%20.20s%d  ", "size of double=", sizeof doubleValue);
    printf("doubleValue=%25.20lf\n", doubleValue);
    printf("%20.20s%d  ", "size of float=", sizeof floatValue);
    printf("floatValue=%25.20f\n", floatValue);
    printf("%20.20s%d  ", "size of int=", sizeof intValue);
    printf("intValue=%d\n", intValue);
    printf("%20.20s%d  ", "size of short int=", sizeof shortIntValue);
    printf("shortIntValue=%d\n", shortIntValue);
    printf("%20.20s%d  ", "size of char=", sizeof charValue);
    printf("charValue=%d\n", charValue);
    return 0;
}
```

2.15.2.

```
#include <stdio.h>

int main() {
    double number;
```

```
    printf("Mennyi legyen a szam? "); scanf("%lf", &number);
    printf("Double-float: %lf\n", number - (float)number);
    printf("Double-int: %lf\n", number - (int)number);
    printf("Double-short: %.2lf\n", number - (short)number);
    printf("Double-char: %.2lf\n", number - (char)number);
    return 0;
}
```

2.16.1.

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

int main() {
    int gotMoney, gotMood, noTime, veryInterested;

    printf("Type 1 for yes and 0 for no!\n");
    printf("gotMoney=");
    scanf("%d", &gotMoney);
    printf("gotMood=");
    scanf("%d", &gotMood);
    printf("noTime=");
    scanf("%d", &noTime);
    printf("veryInterested=");
    scanf("%d", &veryInterested);
    if ( (gotMoney && gotMood && !noTime) || (gotMoney && veryInterested)
|| (gotMood && veryInterested) || (!noTime && veryInterested) ) {
        printf("I will go to holiday!\n");
    } else {
        printf("I will not go to holiday!\n");
    }
    return 0;
}
```

2.16.2.

```
#include <stdio.h>
int main() {
    int isRaining, comeOthers, gotFreeTime;

    printf("Type 1 for yes and 0 for no!\n");
    printf("isRaining: ");
    scanf("%d", &isRaining);
    printf("comeOthers: ");
    scanf("%d", &comeOthers);
    printf("gotFreeTime: ");
    scanf("%d", &gotFreeTime);
    if( (!(isRaining || !gotFreeTime)) || (!(!isRaining || comeOthers)) )
        printf("I'm going hiking!\n");
    else
        printf("Peharps next time.\n");
    return 0;
}
```

2.17.1.

```
#include <stdio.h>

int main() {
    float temperature;

    printf("temperature of water=");
```

```
    scanf("%f", &temperature);
    if (temperature < 0) {
        printf("Solid water\n");
    } else if (temperature < 100) {
        printf("Liquid water\n");
    } else {
        printf("Gaseous water\n");
    }
    return 0;
}
```

2.17.2.

```
#include <stdio.h>
float changer(float);
int main() {
    float celsius;
    float fahrenheit;

    printf("temperature of water(in fahrenheit): ");
    scanf("%f",&fahrenheit);
    celsius = changer(fahrenheit);
    if (celsius < 0)
        printf("The water is solid now.\n");
    else if (celsius > 100)
        printf("The water is gaseous  now.\n");
    else
        printf("The water is liquid now.\n");
    return 0;
}
float changer(float temp) {
    return temp = (temp-32) * 5 / 9;
}
```

2.18.1.

```
#include <stdio.h>
int main() {
    char num;
    printf("Roman to arabic number converter");
    printf("\nGive me a roman number: ");
    scanf("%c", &num);
    if (num > 'Z') num = num - ('a'-'A');
    switch (num) {
        case 'I':   printf("\n1"); break;
        case 'V':   printf("\n5"); break;
        case 'X':   printf("\n10"); break;
        case 'L':   printf("\n50"); break;
        case 'C':   printf("\n100"); break;
        case 'D':   printf("\n500"); break;
        case 'M':   printf("\n1000"); break;
        default:    printf("\nWrong character");
    }
    return 0;
}
```

2.18.2.

```
#include <stdio.h>
#include <ctype.h>
int main() {
    char roman[30];
```

```
    int arabian = 0, size, idxI;

    printf("Roman number:  ");
    scanf("%s", roman);
    size = strlen(roman);
    for (idxI = 0; idxI < size; idxI++) {
        roman[idxI] = toupper(roman[idxI]);
    }
    for (idxI = 0; idxI < size; idxI++) {
        switch (roman[idxI]) {
        case 'I':
            if (roman[idxI + 1] == 'X') {
                arabian += 9;
                idxI += 1;
            }
            else {
                if (roman[idxI + 1] == 'V') {
                    arabian += 4;
                    idxI += 1;
                }
                else {
                    arabian += 1;
                }
            }
            break;
        case 'V':
            arabian += 5;
            break;
        case 'X':
            if (roman[idxI + 1] == 'C') {
                arabian += 90;
                idxI += 1;
            }
            else {
                if (roman[idxI + 1] == 'L') {
                    arabian += 40;
                    idxI += 1;
                }
                else {
                    arabian += 10;
                }
            }
            break;
        case 'L':
            arabian += 50;
            break;
        case 'C':

            if (roman[idxI + 1] == 'M') {
                arabian += 900;
                idxI += 1;
            }
            else {
                if (roman[idxI + 1] == 'D') {
                    arabian += 400;
                    idxI += 1;
                }
                else {
                    arabian += 100;
```

```
                }
            }
            break;
        case 'D':
            arabian += 500;
            break;
        case 'M':
            arabian += 1000;
            break;
        }
    }
    printf("%d\n", arabian);
    return 0;
}
```

2.18.3.

```
#include <stdio.h>
int main() {
    char roman[20];
    int arabian, idxI = 0;
    printf("Arabianian number(1-4999): ");
    scanf("%d", &arabian);
    while (arabian > 0) {
        while ((arabian - 1000) >= 0) {
            roman[idxI++] = 'M';
            arabian -= 1000;
        }
        while ((arabian - 900) >= 0) {
            roman[idxI++] = 'C';
            roman[idxI++] = 'M';
            arabian -= 900;
        }
        while ((arabian - 500) >= 0) {
            roman[idxI++] = 'D';
            arabian -= 500;
        }
        while ((arabian - 400) >= 0) {
            roman[idxI++] = 'C';
            roman[idxI++] = 'D';
            arabian -= 400;
        }
        while ((arabian - 100) >= 0) {
            roman[idxI++] = 'C';
            arabian -= 100;
        }
        while ((arabian - 90) >= 0) {
            roman[idxI++] = 'X';
            roman[idxI++] = 'C';
            arabian -= 90;
        }
        while ((arabian - 50) >= 0) {
            roman[idxI++] = 'L';
            arabian -= 50;
        }
        while ((arabian - 40) >= 0) {
            roman[idxI++] = 'X';
            roman[idxI++] = 'L';
            arabian -= 40;
        }
```

```
        while ((arabian - 10) >= 0) {
            roman[idxI++] = 'X';
            arabian -= 10;
        }
        while ((arabian - 9) >= 0) {
            roman[idxI++] = 'I';
            roman[idxI++] = 'X';
            arabian -= 9;
        }
        while ((arabian - 5) >= 0) {
            roman[idxI++] = 'V';
            arabian -= 5;
        }
        while ((arabian - 4) >= 0) {
            roman[idxI++] = 'I';
            roman[idxI++] = 'V';
            arabian -= 4;
        }
        while ((arabian - 1) >= 0) {
            roman[idxI++] = 'I';
            arabian -= 1;
        }
    }
    roman[idxI++] = '\0';
    printf("%s\n", roman);
    return 0;
}
```

2.19.1.

```
#include <stdio.h>
int main() {
    int idxI, idxJ, size=15;     // egesz valtozo letrehozasa
    for(idxI = 1; idxI <= size; idxI++) {
        for(idxJ = 1; idxJ <= idxI; idxJ++) {
            printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

2.19.2.

```
#include <stdio.h>
int main() {
    int idxI, idxJ, size = 6;
    for (idxI = 0; idxI < size; ++idxI) {
        for (idxJ = 1; idxJ <= size; ++idxJ) {
            if (idxJ < size -idxI)
                printf("");
            else
                printf("*");
        }
        printf("\n");
    }
    return 0;
}
```

2.20.1.

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

```
int main() {
    // square falling off, it is supposed that the console is 80*25
character big
    // for quick computers set bigger size
    int idxA, idxB, idxC, size=600;
    int idxI, idxJ, idxK, drop=22, consoleSize=25;

    for (idxI=0; idxI<drop; idxI++) {
        // erase console
        for (idxJ=0; idxJ<consoleSize; idxJ++) {
            printf("\n");
        }
        // position square vertically 1
        for (idxK=0; idxK<idxI; idxK++) {
            printf("\n");
        }
        // print square
        printf("            ***\n");
        printf("            ***\n");
        printf("            ***\n");
        // position square vertically 2
        for (idxK=0; idxK<consoleSize-idxI-3; idxK++) {
            printf("\n");
        }
        _flushall();
        // delay
        for(idxA = 0; idxA <= size; idxA++) {
            for(idxB = 1; idxB <= size; idxB++) {
                for(idxC = 1; idxC <= size; idxC++) {
                }
            }
        }
    }
    return 0;
}
```

2.20.2.

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

void wait(int seconds) {
    clock_t endwait;
    endwait = clock() + seconds * CLOCKS_PER_SEC;
    while (clock() < endwait) {
    }
}

int main() {
    int idxI, idxJ = 0, size = 0;
    while (1) {
        system("clear");
        size = 4;
        if (idxJ == 25)
            idxJ = 0;
        if (idxJ >= 23) {
            for (idxI = idxJ; idxI >= 23; idxI--) {
                printf("***\n");
                size -= 1;
```

```
            }
        }
        for (idxI = 1; idxI <= idxJ; idxI++)
            printf("\n");
        for (idxI = 1; idxI < size; idxI++)
            printf("***\n");
        wait(1); //ez a mozgas sebessege
        idxJ++;
    }
    return 0;
}
```

2.20.3.

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

void wait(int seconds) { //kesleltetes megvalositasa
    clock_t endwait;
    endwait = clock() + seconds * CLOCKS_PER_SEC;
    while (clock() < endwait) ;
}

int main(){
    int idxI, idxJ, row = 0, direction;
    while(1) { //vegtelen ciklus
         system("clear"); //kepernyo torles
        for (idxI = 1; idxI <= row; idxI++) //ures sorok kirajzolasa
            printf("\n");
        for (idxJ = 0; idxJ < 3; idxJ++) { //szokozok rajzolasa
            for (idxI = 2 - idxJ; idxI > 0; idxI--)
                printf("");
            for (idxI = 0; idxI <= 2*idxJ; idxI++) //csilllagok rajzolasa
                printf("*");
            printf("\n");
        }
         //kesz van a haromszog
        wait(1); //ez a mozgas sebessege
        if (row == 22) //iranyvaltasok
             direction = 0;
         else if (row == 0)
             direction = 1;
        if (direction)
             row++;
         else
             row--;
    }
    return 0;
}
```

2.20.4.

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

void wait(int seconds) {
    clock_t endwait;
    endwait = clock() + seconds * CLOCKS_PER_SEC;
    while (clock() < endwait) ;
```

```
}

int main() {
    int idxI, idxJ = 0, direction;
     char string [] = "*****";
    while (1) {
         system("clear");
        for (idxI = 1; idxI < 4; idxI++) //teglalap kirajzolasa, mindig
eggyel elcsusztatva
           printf("%*.5s\n", idxJ + 5, string);
        wait(1); //ez a mozgas sebessege
        if (idxJ == 75) //iranyvaltas
                 direction = 0;
         else if (idxJ == 0)
             direction=1;

        if (direction)
            idxJ++;
         else
             idxJ--;
    }
    return 0;
}
```

2.21.1.

```
#include <stdio.h>
int main() {
    int x, y; // x=30; y=42 ->6
    printf("Give me the first number: ");
    scanf("%d",&x);
    printf("Give me the second number: ");
    scanf("%d",&y);
    while (x!=y) {
        if (x>y)
               x=x-y;
        else
            y=y-x;
    }
    printf("The biggest common denominator: %d\n", x);
    return 0;
}
```

2.21.2.

```
#include <stdio.h>
int biggest_denominator(int, int);
int main() {
    int x, y, z; //elso, masodik, harmadik szam
    int first, second;
    printf("Give me the first number: ");
    scanf("%d", &x);
    printf("Give me the second number: ");
    scanf("%d", &y);
    printf("Give me the third number: ");
    scanf("%d", &z);
    //for the first 2 number
    first = biggest_denominator(x, y);
    //for the second 2 number
    second = biggest_denominator(y, z);
```

```
    printf("The biggest common denominator: %d\n",
biggest_denominator(first, second)); // compute the result of the fisrt
and second denominations
    return 0;
}

int biggest_denominator(int x, int y) {
    while (x!=y) {
        if (x>y)
            x=x-y;
        else
            y=y-x;
    }
    return x;
}
```

2.22.1.

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    int idxI;
    for (idxI=argc-1; idxI>=0; idxI--) {
        printf("The %d. parameter is: %s\n", idxI, argv[idxI]);
    }
    return 0;
}
```

2.22.2.

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    int idxI = argc-1;
    do {
        printf("The %d. parameter is: %s\n", idxI, argv[idxI]);
        idxI--;
    } while (idxI>=0);
    return 0;
}
```

2.23.1.

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    int idxI=0;
    float x, y;
    do {
        printf("x=");
        scanf("%f", &x);
        printf("y=");
        scanf("%f", &y);
        if ( x>100 || -x>100) {
            printf("x is too big!\n");
            continue;
        }
        if (!y) { // y==0
            printf("y equals 0!\n");
            break;
        }
        printf("%f/%f=%f\n\n", x, y, x/y);
        idxI++;
        if (idxI==3)
            break;
```

```
    } while (1);
    return 0;
}
```

2.24.1.

```
#include <stdio.h>

float deriv(int whichDerivate, float where) {
    // 3x^3-2x^2+6x-1
    float result;

    switch (whichDerivate) {
        case 1:    // 9x^2-4x+6
            result = 9*where*where-4*where+6;
            break;
        case 2:    // 18x-4
            result = 18*where-4;
            break;
        case 3:    // 18
            result = 18;
            break;
        default:
            result = 0;
            printf("\nFirst parameter (%d) is wrong, it should be in 1, 2,
or 3\n", whichDerivate);
            break;
    }
    return result;
}
int main() {
    int idxI;
    float where, rez;

    printf("Where do you want to calculate the derivate: ");
    scanf("%f", &where);
    for (idxI=1; idxI<5; idxI++) {
        rez=deriv(idxI, where);
        printf("The %d. derivative of 9x^2-4x+6 at %f is: %f\n", idxI,
where, rez);
    }
    return 0;
}
```

2.24.2.

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

float deriv(int whichDerivate, float posX) {
    // sin(x)
    float result;

    switch (whichDerivate%4) {
        case 0:    // sin(x)
            result = sin(posX);
            break;
        case 1:    // cos(x)
            result = cos(posX);
            break;
        case 2:    // -sin(x)
```

```
            result = -sin(posX);
            break;
        case 3:    // -cos(x)
            result = -cos(posX);
            break;

        default:
            result = 0;
            printf("\nError!\n");
            break;
    }
    return result;
}
int main() {
    int idxI;
    float posX, result;
    int times;

    printf("Where do you want to calculate the derivate of sin(x) (in
radian): ");
    scanf("%f", &posX);
    printf("How many times? ");
    scanf("%d", &times);
    for (idxI = 1; idxI <= times; idxI++) {
        result = deriv(idxI, posX);
        printf("The %d. derivative of sin(x) at %f is: %f\n", idxI, posX,
result);
    }
    return 0;
}
```

2.24.3.

```
#include <stdio.h>
float deriv(float posX, float poly4, float poly3, float poly2, float
poly1, float poly0) {
    return ( 4*poly4*pow(posX,3) + 3*poly3*pow(posX,2) +
2*poly2*pow(posX,1) + poly1 );
}
int main() {
    float posX, poly4, poly3, poly2, poly1, poly0; //a hely ahol
serivalni szeretnek, és a polinomok egyutthatoi

    printf("What function do you want to derivate?\n");
    printf("x^4: "); scanf("%f", &poly4);
    printf("x^3: "); scanf("%f", &poly3);
    printf("x^2: "); scanf("%f", &poly2);
    printf("x^1: "); scanf("%f", &poly1);
    printf("constant: "); scanf("%f", &poly0);

    printf("So, the ");
    if (poly4) printf("%f x^4 ", poly4);
    if (poly3) printf("%+f x^3 ", poly3);
    if (poly2) printf("%+f x^2 ", poly2);
    if (poly1) printf("%+f x^1 ", poly1);
    if (poly0) printf("%+f ", poly0);
    printf("will be derivated.\n");

    printf("Where do you want to calculate the derivate: ");
    scanf("%f", &posX);
```

```
    printf("The first derivative of the function at %f is: %f\n", posX,
deriv(posX, poly4, poly3, poly2, poly1, poly0));
    return 0;
}
```

2.25.1.

```
#include <stdio.h>

int sum(int data) {
    static int result;
    result += data;
    return result;
}
int main() {
    int num, total;
    char more='y';

    do {
        printf("Provide the next number: ");
        scanf("%d", &num);
        total = sum(num);
        printf("Are there more number: (y/n) ");
        _flushall();
        scanf("%c", &more);
    } while (more=='y');
    printf("The The total is %d.\n", total);
    return 0;
}
```

2.25.2.

```
void cancel() {
    int act;
    act = sum(0);
    sum(-act);
}
```

2.25.3.

```
#include <stdio.h>
int sum(int data, int subTotal) {
    subTotal += data;
    return subTotal;
}
int main() {
    int num, total, subTotal=0;
    char more = 'y';

    do {
        printf("Provide the next number: ");
        scanf("%d", &num);
        subTotal = sum(num, subTotal);
        printf("Are there more number: (y/n) ");
        _flushall();
        scanf("%c", &more);
    } while (more == 'y');
    total = subTotal;
    printf("The total is %d.\n", total);
    return 0;
}
```

2.26.1.

```
#include <stdio.h>

int main() {
    int side;
    int *side1=&side, *side2=&side, *side3=&side;
    int surface, volume;

    printf("The side of the cube is: ");
    scanf("%d", &side);
    surface = 6**side1*(*side2); //* as indirection has big precedence
    volume = *side1*(*side2)**side3;
    printf("The volume of the cube is: %d\n", volume);
    printf("The surface of the cube is: %d\n", surface);
    return 0;
}
```

2.26.2.

```
printf("The volume of the cube is: %d\n", *&*&volume);
```

2.26.3.

```
#include <stdio.h>
#define _USE_MATH_DEFINES
#include <math.h>

int main() {
    float radius;
    float *radius1 = &radius, *radius2 = &radius, *radius3 = &radius;
    float surface, volume;

    printf("The radius of the ball is: ");
    scanf("%f", &radius);
    surface = 4 * (*radius1) * (*radius2) * (*radius3) * (M_PI) / 3;
    volume = 4 * (*radius1) * (*radius2) * (M_PI);
    printf("The volume of the ball is: %f\n", volume);
    printf("The surface of the ball is: %f\n", surface);
    return 0;
}
```

2.27.1

```
#include <stdio.h>

int main() {
    int data=42, code=666, temp;
    int *realAddress=&data, *changedAddress=0, *tryAddress;

    changedAddress = realAddress - code;

    printf("The 1st key: %p\n", changedAddress);
    printf("Press 1 if the other person arrived: ");
    scanf("%d", &temp);
    printf("The 2nd key: %d\n", code);
    changedAddress = NULL;
    code = 0;
    printf("To retrieve the data provide the 1st key: ");
    scanf("%p", &changedAddress);
    printf("Provide the 2nd key: ");
    scanf("%d", &code);
    tryAddress = changedAddress + code;
```

```
    if (tryAddress==realAddress)
        printf("Access granted, the data is: %d\n", *tryAddress);
    else
        printf("Access denied!\n");
    return 0;
}
```

2.28.1.

```
#include <stdio.h>

int main() {
    int data=42, *pData=&data, **ppData=&pData, ***pppData=&ppData;
    int idxI;
    printf("%10s %#p     %#p     %#p     %#p     \n", "address: ",
&pppData, &ppData, &pData, &data);
    printf("%10s ", "");
    for (idxI=0; idxI<4; idxI++)
        printf("------------   ");
    printf("\n%10s |%#p|   |%#p|   |%#p|   |%10d|   \n", "value:",
pppData, ppData, pData, data);
    printf("%10s ", "");
    for (idxI=0; idxI<4; idxI++)
        printf("------------   ");
    printf("\n%10s %-15s%-15s%-15s%-15s", "variable:", "pppData",
"ppData", "pData", "data");
    printf("\n%10s %-15s%-15s%-15s%-15s", "", "", "*pppData", "*ppData",
"*pData");
    printf("\n%10s %-15s%-15s%-15s%-15s", "", "", "", "**pppData",
"**ppData");
    printf("\n%10s %-15s%-15s%-15s%-15s", "", "", "", "", "***pppData");
    return 0;
}
```

2.28.2

```
#include <stdio.h>

int main() {
    double** myMatrix;
    int *pData=&myMatrix, **ppData=&pData, ***pppData=&ppData;
    int idxI, idxJ, idxK, temp = 0;

    //dinamikus memóriafoglalás
    myMatrix = (double**)malloc(2 * sizeof(double*));
    for (idxI=0; idxI<2; idxI++) {
        myMatrix[idxI] = (double*)malloc(3 * sizeof(double));
    }
    for(idxI = 0; idxI < 2; idxI++)
        for(idxJ = 0; idxJ < 3; idxJ++)
            myMatrix[idxI][idxJ] = ++temp;
    printf("****Adresses****\n");
    printf("myMatrix**: [%#p] -> myMatrix[0,1]*: [%#p, %#p]\n",
&myMatrix, &myMatrix[0], &myMatrix[1]);
    printf("myMatrix[0]*: [%#p]-> myMatrix[0][0,1,2]: [%#p, %#p, %#p]\n",
&myMatrix[0], &myMatrix[0][0], &myMatrix[0][1], &myMatrix[0][2]);
    printf("myMatrix[1]*: [%#p]-> myMatrix[1][0,1,2]: [%#p, %#p, %#p]\n",
&myMatrix[1], &myMatrix[1][0], &myMatrix[1][1], &myMatrix[1][2]);
    printf("\n***Values****\n");
    printf("myMatrix**: [%#p] : myMatrix[0,1]*: [%#p, %#p]\n", myMatrix,
myMatrix[0], myMatrix[1]);
```

```
    printf("myMatrix[0]*: [%#p] : myMatrix[0][0,1,2]: [%.2lf, %.2lf,
%.2lf]\n", myMatrix[0], myMatrix[0][0], myMatrix[0][1], myMatrix[0][2]);
    printf("myMatrix[1]*: [%#p] : myMatrix[1][0,1,2]: [%.2lf, %.2lf,
%.2lf]\n", myMatrix[1], myMatrix[1][0], myMatrix[1][1], myMatrix[1][2]);

    for (idxI=0; idxI<2; idxI++) //felszabaditás
        free(myMatrix[idxI]);
    free(myMatrix);
    return 0;
}
```

2.29.1.

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

int main() {
    int choice;
    short int *si1=NULL, *si2=NULL;
    long double *ld1=NULL, *ld2=NULL;
    printf("Short int operands (1)\n");
    printf("Long double operands (2)\n");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            si1 = (short int*)malloc(1*sizeof(short int));
            si2 = (short int*)malloc(1*sizeof(short int));
            printf("si1=");
            scanf("%hd", si1);
            printf("si2=");
            scanf("%hd", si2);
            *si1 = *si1 + * si2;
            printf("The increased value is: %hd\n", *si1);
            free(si1); si1=NULL;
            free(si2); si2=NULL;
            break;
        case 2:
            ld1 = (long double*)malloc(1*sizeof(long double));
            ld2 = (long double*)malloc(1*sizeof(long double));
            printf("ld1=");
            scanf("%Lf", ld1);
            printf("ld2=");
            scanf("%Lf", ld2);
            *ld1 = *ld1 + *ld2;
            printf("The increased value is: %Lf\n", *ld1);
            free(ld1); ld1=NULL;
            free(ld2); ld2=NULL;
            break;
    }
    return 0;
}
```

2.30.1.

```
#include <stdio.h>
struct Var {
    char varChar;
    short int varShortInt;
    int varInt;
    float varFloat;
    double varDouble;
```

```
    long double varLongDouble;
    int *varPInt;
};

int main() {
    struct Var myVar;
    printf("varChar=");
    scanf("%c", &myVar.varChar);
    printf("varShortInt=");
    scanf("%hd", &myVar.varShortInt);
    printf("varInt=");
    scanf("%d", &myVar.varInt);
    printf("varFloat=");
    scanf("%f", &myVar.varFloat);
    printf("varDouble=");
    scanf("%lf", &myVar.varDouble);
    printf("varLongDouble=");
    scanf("%Lf", &myVar.varLongDouble);
    printf("varPInt=");
    scanf("%p", &myVar.varPInt);

    printf("\nmyVar\n");
    printf("varChar=%c\n", myVar.varChar);
    printf("varShortInt=%hd\n", myVar.varShortInt);
    printf("varInt=%d\n", myVar.varInt);
    printf("varFloat=%g\n", myVar.varFloat);
    printf("varDouble=%lg\n", myVar.varDouble);
    printf("varLongDouble=%Lg\n", myVar.varLongDouble);
    printf("varPInt=%p", myVar.varPInt);
    return 0;
}
```

2.30.2.

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

struct Var {
    char varChar;
    short int varShortInt;
    int varInt;
    float varFloat;
    double varDouble;
    long double varLongDouble;
    int *varPInt;
};
typedef struct Var VarType;

void readVarType(VarType *myVar) {
    printf("varChar=");
    scanf("%c", &myVar->varChar);
    printf("varShortInt=");
    scanf("%hd", &myVar->varShortInt);
    printf("varInt=");
    scanf("%d", &myVar->varInt);
    printf("varFloat=");
    scanf("%f", &myVar->varFloat);
    printf("varDouble=");
    scanf("%lf", &myVar->varDouble);
    printf("varLongDouble=");
```

```
    scanf("%Lf", &myVar->varLongDouble);
    myVar->varPInt = &myVar->varInt;
}

void printVarType(VarType myVar) {
    printf("\nmyVar\n");
    printf("varChar=%c\n", myVar.varChar);
    printf("varShortInt=%hd\n", myVar.varShortInt);
    printf("varInt=%d\n", myVar.varInt);
    printf("varFloat=%g\n", myVar.varFloat);
    printf("varDouble=%lg\n", myVar.varDouble);
    printf("varLongDouble=%Lg\n", myVar.varLongDouble);
    printf("*varPInt=%d", *myVar.varPInt);
}

int main() {
    VarType myVar, *copy=NULL;
    readVarType(&myVar);
    printVarType(myVar);
    copy = (VarType*)malloc(sizeof(VarType)*1);
    *copy = myVar;
    printVarType(*copy);
    *((*copy).varPInt)=666; // *(*copy).varPInt=666; *copy->varPInt=666;
    printVarType(myVar);
    free(copy); copy=NULL;
    return 0;
}
```

2.31.1.

```
#include <stdio.h>
#include <malloc.h>
typedef struct {
    char name[200];
    int age;
    int ID;
} Person;

typedef struct {
    int carat;
    float price;
    int numberOfStones;
} Ring;

typedef struct {
    Person man, wife;
    Ring weddingRing, engagementRing;
    int guestNumber;
    Person *guests;
} Wedding;

void printPerson(Person myPerson) {
    printf("person{");
    printf("name=%s, ", myPerson.name);
    printf("age=%d, ", myPerson.age);
    printf("ID=%d}", myPerson.ID);
}

void printRing(Ring myRing) {
    printf("ring{");
```

```
    printf("carat=%d, ", myRing.carat);
    printf("price=%f, ", myRing.price);
    printf("number of stones=%d}", myRing.numberOfStones);
}

void printWedding(Wedding myWedding) {
    int idxI;
    printf("wedding{\n\tman=");
    printPerson(myWedding.man);
    printf(",\n\twife=");
    printPerson(myWedding.wife);
    printf(",\n\twedding ring=");
    printRing(myWedding.weddingRing);
    printf(",\n\tengagement ring=");
    printRing(myWedding.engagementRing);
    printf(",\n\tGuests={");
    for (idxI=0; idxI<myWedding.guestNumber; idxI++) {
        printf("\n\t\t");
        printPerson(myWedding.guests[idxI]);
        printf(",");
    }
    printf("\n\t}\n}");
}

int main() {
    const int friendNumber=3;
    Person friends[friendNumber]={{"Jani", 28, 12345}, {"Emese", 29,
66666}, {"Zoli", 32, 222222}};
    Wedding brotherWedding={
        {"Arpi", 32, 23332},
        {"Panna", 29, 22222},
        {16, 60000, 0},
        {24, 200000, 3},
        friendNumber, friends
    };
    printf("brotherWedding=");
    printWedding(brotherWedding);
    return 0;
}
```

2.32.1.

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
typedef struct {
    char manufacturer[20]; // static
    int productNumber;
    char* description; // dynamic
    float price;
} Fan;
Fan readFan() {
    Fan rec;
    int descriptionLength;
    printf("Provide the data of the fan!\n");
    printf("manufacturer: ");
    gets(rec.manufacturer);
    printf("product number: ");
    scanf("%d", &rec.productNumber);
    printf("max description length: ");
```

```
    scanf("%d", &descriptionLength);
    rec.description = (char*)malloc(sizeof(char)*descriptionLength);
    _flushall();
    printf("description: ");
    gets(rec.description);
    printf("price:");
    scanf("%f", &rec.price);
    //. prefereniája nagyobb a mint az &-e
    return rec;
}
void printFan(Fan rec) {
    printf("Fan data\n");
    printf("manufacturer: %s\n", rec.manufacturer);
    printf("product number:%d\n", rec.productNumber);
    printf("description:%s\n", rec.description);
    printf("price:%5.2f HUF\n", rec.price);
}
int main() {
    Fan myFan;
    myFan = readFan();
    printFan(myFan);
    free(myFan.description); myFan.description=NULL;
    return 0;
}
```

2.32.2.

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

typedef struct {
    char manufacturer[20]; // static
    int productNumber;
    char* description; // dynamic
    float price;
} Fan;

Fan readFan() {
    Fan rec;
    int descriptionLength;

    printf("Provide the data of the fan!\n");
    printf("manufacturer: ");
    scanf("%s", rec.manufacturer);
    printf("product number: ");
    scanf("%d", &rec.productNumber);
    printf("max description length: ");
    scanf("%d", &descriptionLength);
    rec.description = (char*) malloc(sizeof(char) * descriptionLength);
    printf("description: ");
    scanf("%s", rec.description);
    printf("price: ");
    scanf("%f", &rec.price);
    //. prefereniája nagyobb a mint az &-e
    return rec;
}

void printFan(Fan rec) {
    printf("manufacturer: %s\n", rec.manufacturer);
```

```
    printf("product number:%d\n", rec.productNumber);
    printf("description:%s\n", rec.description);
    printf("price:%5.2f HUF\n", rec.price);
}

int main() {
    Fan myFan,myFan2;
    myFan = readFan();
    myFan2=myFan;
    myFan.description="A new file";
    printf("Fan data\n");
    printFan(myFan);
    printf("Fan2 data\n");
    printFan(myFan2);
    free(myFan.description);
    myFan.description = NULL;
    free(myFan2.description);
    myFan2.description = NULL;

    return 0;
}
```

2.33.1.

```
#include <stdio.h>

const int maxStudentNumber=100;
const int maxCodeLength=6;

typedef struct {
    int sNum;
    char sCode[maxStudentNumber][maxCodeLength+1];
    float sPoints[maxStudentNumber];
} Class;

void printClass(Class *myClass, char comments[4][3][40]) {
    int idxI;
    printf("%+10s%#20p%#20p%#20p\n", "address: ", myClass, myClass-
>sCode, myClass->sPoints);
    printf("%+10s%20d%20c%20.1f\n", "value: ", myClass->sNum, myClass-
>sCode[0][0], myClass->sPoints[0]);
    for (idxI=0; idxI<4; idxI++) {
       printf("%10s%20s%20s%20s\n", "", comments[idxI][0],
comments[idxI][1], comments[idxI][2]);
    }
}

int main() {
    Class myCl[2]={
        {3, {"AA34", "BE3E", "CLY56"}, {16.4, 23.2, 40.0}},
        {2, {"NMA444", "SE63"}, {13.2, 50.0}},
    };
    char myComments[4][3][40]= {
        {"myCl[0] "},
        {"myCl[0].sNum", "myCl[0].sCode", "myCl[0].sPoints"},
        {"",                          "myCl[0].sCode[0]",
"myCl[0].sPoints[0]"},
        {"",                          "myCl[0].sCode[0][0]"}
    };
```

```
    printClass(&myCl[0], myComments);
    printf("\n\n");
    myComments[0][0][5] = '1';
    myComments[1][0][5] = '1';
    myComments[1][1][5] = '1';
    myComments[1][2][5] = '1';
    myComments[2][1][5] = '1';
    myComments[2][2][5] = '1';
    myComments[3][1][5] = '1';
    printClass(&myCl[1], myComments);
    return 0;
}
```

2.34.1.

```
#include <stdio.h>
const int maxStudentNumber=100;
const int maxCodeLength=6;

typedef struct {
    char name[100];
    unsigned int sex :1; // 0 - female, 1 - male
    unsigned int grownUp :1; // 0 - no, 1 - yes
    unsigned int zodiac :6; // 0 - Aries, 1 - Taurus, 2 - Gemini, 3 -
Cancer, 4 - Leo
    // 5 - Virgo, 6- Libra, 7 - Scorpio, 8 - Sagittarius, 9 - Capricorn,
10 - Aquarious
    // 11 - Pisces
    unsigned int religion :3; // 0 - Judaism, 1 - Christian, 2 - Islam, 3
- Hinduism, 4 - Budhism, 5 - other
    unsigned int bloodType :2; // 0 - 0, 1 - A, 2 - B, 3 - AB
    unsigned int RH :1; // 0 - -, 1 - +
} Person;

void readPerson(Person *myPerson) {
    int dummy;
    printf("name=");
    scanf("%s", myPerson->name);

    printf("sex: 0 - female, 1 - male ");
    scanf("%d", &dummy);
    myPerson->sex=dummy;

    printf("grown up: 0 - no, 1 - yes ");
    scanf("%d", &dummy);
    myPerson->grownUp=dummy;

    printf("zodiac: 0 - Aries, 1 - Taurus, 2 - Gemini, 3 - Cancer, 4 -
Leo, 5 - Virgo, \
    6 - Libra, 7 - Scorpio, 8 - Sagittarius, 9 - Capricorn, 10 -
Aquarious, 11 - Pisces ");
    scanf("%d", &dummy);
    myPerson->zodiac=dummy;

    printf("religion: 0 - Judaism, 1 - Christian, 2 - Islam, 3 -
Hinduism, \
    4 - Budhism, 5 - other ");
    scanf("%d", &dummy);
    myPerson->religion=dummy;
```

```
    printf("bloodType: 0 - 0, 1 - A, 2 - B, 3 - AB ");
    scanf("%d", &dummy);
    myPerson->bloodType=dummy;

    printf("RH: 0 - -, 1 - + ");
    scanf("%d", &dummy);
    myPerson->RH=dummy;
}

void printPerson(Person *myPerson) {
    printf("person={name=%s, sex=", myPerson->name);
    switch (myPerson->sex) {
        case 0: printf("female, "); break;
        case 1: printf("male, "); break;
    }
    printf("grown up=");
    switch (myPerson->grownUp) {
        case 0: printf("no, "); break;
        case 1: printf("yes, "); break;
    }
    printf("zodiac=");
    switch (myPerson->zodiac) {
        case 0: printf("Aries, "); break;
        case 1: printf("Taurus, "); break;
        case 2: printf("Gemini, "); break;
        case 3: printf("Cancer, "); break;
        case 4: printf("Leo, "); break;
        case 5: printf("Virgo, "); break;
        case 6: printf("Libra, "); break;
        case 7: printf("Scorpio, "); break;
        case 8: printf("Sagittarius, "); break;
        case 9: printf("Capricorn, "); break;
        case 10: printf("Aquarious, "); break;
        case 11: printf("Pisces, "); break;
    }
    printf("religion=");
    switch (myPerson->religion) {
        case 0: printf("Judaism, "); break;
        case 1: printf("Christian, "); break;
        case 2: printf("Islam, "); break;
        case 3: printf("Hinduism, "); break;
        case 4: printf("Budhism, "); break;
        case 5: printf("other, "); break;
    }
    printf("bloodType=");
    switch (myPerson->bloodType) {
        case 0: printf("0, "); break;
        case 1: printf("A, "); break;
        case 2: printf("B, "); break;
        case 3: printf("AB, "); break;
    }
    printf("RH=");
    switch (myPerson->RH) {
        case 0: printf("-} "); break;
        case 1: printf("+} "); break;
    }
    printf("structure size in bytes = %d\n", sizeof(*myPerson));
}
```

```
int main() {
    Person me;
    readPerson(&me);
    printf("\n");
    printPerson(&me);
    return 0;
}
```

2.34.2.

```
#include <stdio.h>
const int maxStudentNumber = 100;
const int maxCodeLength = 6;

typedef struct {
    char name[100];
    unsigned int sex; // 0 - female, 1 - male
    unsigned int grownUp; // 0 - no, 1 - yes
    unsigned int zodiac; // 0 - Aries, 1 - Taurus, 2 - Gemini, 3 -
Cancer, 4 - Leo
    // 5 - Virgo, 6- Libra, 7 - Scorpio, 8 - Sagittarius, 9 - Capricorn,
10 - Aquarious
    // 11 - Pisces
    unsigned int religion; // 0 - Judaism, 1 - Christian, 2 - Islam, 3 -
Hinduism, 4 - Budhism, 5 - other
    unsigned int bloodType; // 0 - 0, 1 - A, 2 - B, 3 - AB
    unsigned int RH; // 0 - -, 1 - +
} Person;

void readPerson(Person *myPerson) {
    int dummy;
    printf("name=");
    scanf("%s", myPerson->name);

    printf("sex: 0 - female, 1 - male ");
    scanf("%d", &dummy);
    myPerson->sex = dummy;

    printf("grown up: 0 - no, 1 - yes ");
    scanf("%d", &dummy);
    myPerson->grownUp = dummy;

    printf(
           "zodiac: 0 - Aries, 1 - Taurus, 2 - Gemini, 3 - Cancer, 4 -
Leo, 5 - Virgo, \
    6 - Libra, 7 - Scorpio, 8 - Sagittarius, 9 - Capricorn, 10 -
Aquarious, 11 - Pisces ");
    scanf("%d", &dummy);
    myPerson->zodiac = dummy;

    printf(
           "religion: 0 - Judaism, 1 - Christian, 2 - Islam, 3 -
Hinduism, \
    4 - Budhism, 5 - other ");
    scanf("%d", &dummy);
    myPerson->religion = dummy;

    printf("bloodType: 0 - 0, 1 - A, 2 - B, 3 - AB ");
    scanf("%d", &dummy);
    myPerson->bloodType = dummy;
```

```
    printf("RH: 0 - -, 1 - + ");
    scanf("%d", &dummy);
    myPerson->RH = dummy;
}

void printPerson(Person *myPerson) {
    printf("person={name=%s, sex=", myPerson->name);
    switch (myPerson->sex) {
    case 0:
        printf("female, ");
        break;
    case 1:
        printf("male, ");
        break;
    }
    printf("grown up=");
    switch (myPerson->grownUp) {
    case 0:
        printf("no, ");
        break;
    case 1:
        printf("yes, ");
        break;
    }
    printf("zodiac=");
    switch (myPerson->zodiac) {
    case 0:
        printf("Aries, ");
        break;
    case 1:
        printf("Taurus, ");
        break;
    case 2:
        printf("Gemini, ");
        break;
    case 3:
        printf("Cancer, ");
        break;
    case 4:
        printf("Leo, ");
        break;
    case 5:
        printf("Virgo, ");
        break;
    case 6:
        printf("Libra, ");
        break;
    case 7:
        printf("Scorpio, ");
        break;
    case 8:
        printf("Sagittarius, ");
        break;
    case 9:
        printf("Capricorn, ");
        break;
    case 10:
        printf("Aquarious, ");
```

```
        break;
    case 11:
        printf("Pisces, ");
        break;
    }
    printf("religion=");
    switch (myPerson->religion) {
    case 0:
        printf("Judaism, ");
        break;
    case 1:
        printf("Christian, ");
        break;
    case 2:
        printf("Islam, ");
        break;
    case 3:
        printf("Hinduism, ");
        break;
    case 4:
        printf("Budhism, ");
        break;
    case 5:
        printf("other, ");
        break;
    }
    printf("bloodType=");
    switch (myPerson->bloodType) {
    case 0:
        printf("0, ");
        break;
    case 1:
        printf("A, ");
        break;
    case 2:
        printf("B, ");
        break;
    case 3:
        printf("AB, ");
        break;
    }
    printf("RH=");
    switch (myPerson->RH) {
    case 0:
        printf("-}");
        break;
    case 1:
        printf("+}");
        break;
    }
    printf("structure size in bytes = %d\n", sizeof(*myPerson));
}
int main() {
    Person me;
    readPerson(&me);
    printf("\n");
    printPerson(&me);
    return 0;
}
```

2.35.1.

```
#include <stdio.h>

typedef enum {
    awfull, bad, weak, good, excelent
} Result;

void printResult(Result value) {
    switch (value) {
        case awfull: printf("awfull"); break;
        case bad: printf("bad"); break;
        case weak: printf("weak"); break;
        case good: printf("good"); break;
        case excelent: printf("excelent"); break;

    }
}

int main() {
    Result exam1, exam2;
    int dummy;

    printf("Provide the result of your exam:\nawfull - %d, bad - %d, weak
- %d, good- %d, excelent - %d ", awfull, bad, weak, good, excelent);
    scanf("%d", &dummy);
    exam1 = (Result)dummy;
    printf("\nYour result is: ");
    printResult(exam1);
    printf("\n\nProvide the result of your repeating exam:\nawfull - %d,
bad - %d, weak - %d, good- %d, excelent - %d ", awfull, bad, weak, good,
excelent);
    scanf("%d", &dummy);
    exam2 = (Result)dummy;

    if (exam1 < exam2) {
        printf("Very good, you improved!");
    }
    if (exam1 == exam2) {
        printf("At least you tried!");
    }
    if (exam1 > exam2) {
        printf("You must be very unlucky!");
    }
    return 0;
}
```

2.35.2.

define használata esetén, amikor bevezetjük a közepes értékelést, akkor két rossz lehetőség közül választhatunk. Vagy eltoljuk a közepes feletti eredményeket, hogy helyet csináljunk a közepesnek vagy olyan egész értéket rendelünk a közepeshez, amely nagyobb, mint a kiválóhoz tartozó érték.

2.36.1.

```
#include <stdio.h>
#include <io.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <string.h>
```

```
int main() {
    char buffer[50];
    char dosCommand[200] = "echo /* End of file */ >> ";

    printf("file name=");
    scanf("%s", buffer);
    if( _chmod(buffer, _S_IREAD | _S_IREAD) == -1 )
        printf("File not found\n");
    else
        printf("Mode changed to read-only\n");
    strcat(dosCommand, buffer);
    system(dosCommand);

   /* Change back to read/write: */
    if( _chmod(buffer, _S_IWRITE ) == -1)
        printf("File not found\n");
    else
        printf("Mode changed to read/write\n");
    system(dosCommand);
    return 0;
}
```

2.36.2.

```
#include <stdio.h>
#include <io.h>
#include <sys/stat.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char buffer[50];
    char dosCommand[200] = "echo /* End of file */ >> ";

    printf("file name=");
    scanf("%s", buffer);
    strcat(dosCommand, buffer);

    /* Change to read/write: */
    if (_chmod(buffer, _S_IWRITE) == -1)
        printf("File not found\n");
    else {
        printf("Mode changed to read/write\n");
        remove(buffer);
        printf("File deleted\n");
    }
    return 0;
}
```

2.37.1.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
extern int errno;

int main(int argc, char* argv[]) {
    char oldName[50], newName[50];
    int result;
```

```
    if (argc != 3) {
        printf("Usage:\n prgName oldName newName\n");
        exit(1);
    }
    strcpy(oldName, argv[1]);
    strcpy(newName, argv[2]);
    rename(oldName, newName);
    result = errno;
    switch (result) {
        case 0:
            printf("Done! ");
            break;
        case EEXIST:
            printf("%s already exists!", newName);
            break;
        case ENOENT:
            printf("%s does not exist!", oldName);
            break;
        case EINVAL:
            printf("Wrong characters in '%s'!", newName);
            break;

    }
    return 0;
}
```

2.38.1.

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
int main() {
    FILE* tf;
    int idxI, data, min, max;
    char fileName[200];
    double num;

    printf("This program will save certain ammount number random number
in a file.\n");
    printf("file name=");
    scanf("%s", fileName);
    printf("min number=");
    scanf("%d", &min);
    printf("max number=");
    scanf("%d", &max);
    printf("number of data=");
    scanf("%lf", &num);

    srand((unsigned)time(NULL));
    tf=fopen(fileName, "w");
    if (tf==NULL) {
        fprintf(stderr, "File open to write has failed!\n");
        exit(-1);
    }
    for(idxI=0; idxI<(int)num; idxI++) {
        data = rand()+max+1-min;
        fprintf(tf, "%d\n", data);
    }
    fclose(tf); tf = NULL;
    return 0;
```

```
}
```

2.38.2.

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

int main() {
    FILE* tf;
    int idxI, data, min, max, db = 0;
    char fileName[200];
    int num;

    printf("This program will save certain ammount number random number
in a file.\n");
    printf("file name=");
    scanf("%s", fileName);
    printf("min number=");
    scanf("%d", &min);
    printf("max number=");
    scanf("%d", &max);
    printf("number of data=");
    scanf("%d", &num);
    srand((unsigned) time(NULL));
    tf = fopen(fileName, "w");
    if (tf == NULL) {
        fprintf(stderr, "File open to write has failed!\n");
        exit(-1);
    }
    fprintf(tf, "%d\n", num);
    for (idxI = 0; idxI < num; idxI++) {
        data = rand() % (max + 1 - min) + min;
        fprintf(tf, „%d\t”, data);
        db++;
        if (db == 100) {
            fprintf(tf, "\n");
            db = 0;
        }
    }
    fclose(tf);
    tf = NULL;
    return 0;
}
```

2.39.1.

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

double average(char* fileName) {
    double result, sum=0;
    int counter=0, data;
    FILE* file;

    file = fopen(fileName, "r");
    while ( !feof(file) ) {
        fscanf(file, "%d", &data);
        counter++;
        sum += data;
    }
```

```
    fclose(file); file = NULL;
    result = sum / counter;
    return result;
}

int main() {
    FILE *orig=NULL, *even=NULL, *odd=NULL;
    int evenCounter=0, oddCounter=0, data;
    char origFileName[200], evenFileName[200]="even.txt",
oddFileName[200]="odd.txt";
    double evenAverage, oddAverage;

    printf("This program will divide a bunch of number according to
parity.\n");
    printf("file name=");
    scanf("%s", origFileName);
    orig = fopen(origFileName, "r");
    if (orig == NULL) {
        fprintf(stderr, "File open to read has failed!\n");
        exit(-1);
    }
    even = fopen(evenFileName, "w");
    odd = fopen(oddFileName, "w");

    while ( !feof(orig) ) {
        fscanf(orig, "%d", &data);
        if (data%2 == 0) {
            evenCounter++;
            fprintf(even, "%d\n", data);
        } else {
            oddCounter++;
            fprintf(odd, "%d\n", data);
        }
    }
    fclose(orig); orig = NULL;
    fclose(even); even = NULL;
    fclose(odd); odd = NULL;
    printf("There were %d even and %d odd number, their ratio is:
%8lg\n", evenCounter, oddCounter, (double)evenCounter/oddCounter);
    evenAverage = average(evenFileName);
    oddAverage = average(oddFileName);
    printf("The average of the even numbers: %.9lg\nThe average of the
odd numbers: %.9lg\n", evenAverage, oddAverage);
    return 0;
}
```

2.40.1.

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

typedef struct {
    int ID;
    double price;
} Car;

typedef struct {
    char name[200];
    int age;
    int carNumber;
```

```
    Car* myCars;
} Person;

void printHalfTag(FILE* file, int depth, char* tag) {
    int idxI;
    for (idxI=0; idxI<depth; idxI++) {
        fprintf(file, "\t");
    }
    fprintf(file, "<%s>\n", tag);
}

void printFullTag(FILE* file, int depth, char* tag, char* text) {
    int idxI;
    for (idxI=0; idxI<depth; idxI++) {
        fprintf(file, "\t");
    }
    fprintf(file, "<%s>%s</%s>\n", tag, text, tag);
}

void printPerson(FILE* file, Person* buddy) {
    int idxI;
    char temp[500];
    printHalfTag(file, 0, "Person");
    printFullTag(file, 1, "Name", buddy->name);
    sprintf(temp, "%d", buddy->age);
    printFullTag(file, 1, "Age", temp);
    printFullTag(file, 1, "Count", buddy-> carNumber);
    for (idxI=0; idxI<buddy->carNumber; idxI++) {
        printHalfTag(file, 1, "Car");
        sprintf(temp, "%d", buddy->myCars[idxI].ID);
        printFullTag(file, 2, "ID", temp);
        sprintf(temp, "%lf", buddy->myCars[idxI].price);
        printFullTag(file, 2, "price", temp);
        printHalfTag(file, 1, "/Car");
    }
    printHalfTag(file, 0, "/Person");
}

int main() {
    const int buddyNumber=3;
    int idxI;
    Car init[buddyNumber][3]={ { {0, 50}, {2, 12.369}, {21, 0.569} },
                      {{6, 21}, {1, 15.4}},
                      {{3, 5.7}}, };
    Person buddies[buddyNumber]={{"Jani", 22, 3, init[0]}, {"Evi", 36, 2,
init[1]}, {"Zotya", 25, 1, init[2]} };
    FILE *file=NULL;
    char fileName[200]="car.txt";

    file = fopen(fileName, "w");

    for (idxI=0; idxI<buddyNumber; idxI++) {
        printPerson(file, &buddies[idxI]);
    }
    fclose(file); file = NULL;
    return 0;
}
```

2.40.2.

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

typedef struct {
    int ID;
    double price;
} Car;

typedef struct {
    char name[200];
    int age;
    int carNumber;
    Car* myCars;
} Person;

void printHalfTag(FILE* file, int depth, char* tag) {
    int idxI;

    for (idxI = 0; idxI < depth; idxI++) {
        fprintf(file, "\t");
    }
    fprintf(file, "<%s>\n", tag);
}

void printFullTag(FILE* file, int depth, char* tag, char* text) {
    int idxI;

    for (idxI = 0; idxI < depth; idxI++) {
        fprintf(file, "\t");
    }
    fprintf(file, "<%s\"%s\"/>\n", tag, text);
}

void printPerson(FILE* file, Person* buddy) {
    int idxI;
    char temp[500];

    printHalfTag(file, 0, "Person");
    printFullTag(file, 1, "Name value=", buddy->name);
    sprintf(temp, "%d", buddy->age);
    printFullTag(file, 1, "Age value=", temp);
    printHalfTag(file, 1, "Cars");
    for (idxI = 0; idxI < buddy->carNumber; idxI++) {
        printHalfTag(file, 2, "Car");
        sprintf(temp, "%d", buddy->myCars[idxI].ID);
        printFullTag(file, 3, "Id value", temp);
        sprintf(temp, "%lf", buddy->myCars[idxI].price);
        printFullTag(file, 3, "Price value=", temp);
        printHalfTag(file, 2, "/Car");
    }
    printHalfTag(file, 1, "/Cars");
    printHalfTag(file, 0, "/Person");
}

int main() {
    const int buddyNumber = 3;
    int idxI;
```

```
    Car init[3][3] = { { { 0, 50 }, { 2, 12.369 }, { 21, 0.569 } }, {
            { 6, 21 }, { 1, 15.4 }
        }, { { 3, 5.7 } },
    };
    Person buddies[3] = { { "Jani", 22, 3, init[0] },
        { "Evi", 36, 2, init[1] }, { "Zotya", 25, 1, init[2] }
    };
    FILE *file = NULL;
    char fileName[200] = "car.txt";

    file = fopen(fileName, "w");
    for (idxI = 0; idxI < buddyNumber; idxI++) {
        printPerson(file, &buddies[idxI]);
    }
    fclose(file);
    file = NULL;
    printf("XML file created in car.txt.\n");
    return 0;
}
```

2.40.3.

```
// file.xml begins
<Person>
    <Name>Jani</Name>
    <Age>22</Age>
    <Car>
        <ID>0</ID>
        <price>50.000000</price>
    </Car>
    <Car>
        <ID>2</ID>
        <price>12.369000<price>
    </Car>
    <Car>
        <ID>21</ID>
        <price>0.569000</price>
    </Car>
</Person>
<Person>
    <Name>Evi</Name>
    <Age>36</Age>
    <Car>
        <ID>6</ID>
        <price>21.000000</price>
    </Car>
    <Car>
        <ID>1<ID>
        <price>15.400000</price>
    </Car>
</Person>
<Person>
    <Name>Zotya</Name>
    <Age>25</Age>
    <Car>
        <ID>3</ID>
        <price>5.700000</price>
    </Car>
</Person>
// file.xml ends
```

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

int main() {
    char data[200][200];
    char tag[50], newtag[50];
    int idxI, idxJ, idxK, rows = 0, TagIdx = 0, Found = 0,Error=0;
    FILE * fd = fopen("file.xml", "r");
    if (!fd) {
        printf("file.xml is not found");
        exit(1);
    }
    while (!feof(fd)) {
        fscanf(fd, "%s", data[rows]);
        rows++;
    }
    for (idxI = 0; idxI < rows; idxI++) {
        idxJ = 0;
        TagIdx = 0;
        Found = 0;
        if (data[idxI][idxJ] == '<' && data[idxI][idxJ + 1] != '/') {
            do {
                tag[TagIdx] = data[idxI][idxJ];
                TagIdx++;
                idxJ++;
            } while (data[idxI][TagIdx - 1] != '>');
            tag[TagIdx] = '\0';
            newtag[strlen(tag) + 1] = '\0';
            for (idxK = strlen(tag); idxK > 0; idxK--) {
                if (idxK != 1) {
                    newtag[idxK] = tag[idxK - 1];
                } else
                    newtag[idxK] = '/';
            }
            newtag[0] = tag[0];
            if (strlen(data[idxI]) > strlen(tag)) {
                if (strstr(data[idxI], newtag)) {
                    Found = 1;
                    data[idxK][0] = '\0';
                } else {
                    printf("Error at line: %d.\n\tNo close TAG for %s.\n",
idxI + 1,
                            tag);
                    Error=1;
                }
            } else {
                for (idxK = idxI+1; idxK < rows; idxK++) {
                    if (!strcmp(tag, data[idxK]) && !Found) {
                        printf("At line %d.\n\tNo closeTAG for %s.\n",
                                idxI+1,tag);
                        Error=1;
                        idxK=rows;
                    } else if (strstr(data[idxK], newtag) && !Found) {
                        Found = 1;
                        data[idxK][0] = '\0';
```

```
                    }
                }
            }
        }
    }
    if(!Error){
        printf("No errors detected!\n");
    }
    return 0;
}
```

2.41.1.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    clock_t actClock;
    double actSec=0;
    while (actSec < 11) {
        actClock = clock();
        actSec = (double)actClock / CLOCKS_PER_SEC;
        printf("\n%lf", actSec);
    }
    return 0;
}
```

2.41.2.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    clock_t actClock;
    double actSec=0;
    int intActSec=0, intOldSec=0;
    while (actSec < 11) {
        actClock = clock();
        actSec = (double)actClock / CLOCKS_PER_SEC;
        intActSec = (int)actSec;
        if (intActSec>intOldSec) {
            printf("\n%d", intActSec);
            intOldSec = intActSec;
        }
    }
    return 0;
}
```

2.42.1.

```
#include <stdio.h>
#include <sys/timeb.h>
#include <time.h>

int main() {
    struct _timeb timebuffer;
    char* timeline;
    int start, act;

    _ftime(&timebuffer);
    start = timebuffer.time;
    do {
```

```
        _ftime(&timebuffer);
        timeline = ctime(&timebuffer.time);
        act = timebuffer.time;
        printf("\n%.8s.%hu", &timeline[11], timebuffer.millitm);
    } while (act-start < 10);
    return 0;
}
```

2.43.1.

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

int main() {
    clock_t start, end;
    int wait;
    char temp[50];
    double actWait;

    srand((unsigned)time(NULL));
    printf("Press 'a' and enter to start the clock!\n");
    printf("Do the same to stop it!\n");
    printf("Type quit to end program!\n");
    while (strcmp(temp, "quit") != 0) {
        wait = rand()%4 + 1;
        printf("\nWait for %d seconds!\n", wait);
        printf("Begin: ");
        scanf("%s", temp);
        if (strcmp(temp, "quit")==0) break;
        start = clock();
        printf("End: ");
        scanf("%s", temp);
        end = clock();
        actWait = (double)(end - start) / CLOCKS_PER_SEC;
        printf("You have waited %6.2lf seconds instead of %d. ", actWait,
wait);
    }
    return 0;
}
```

2.44.1.

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

int main() {
    int year, month, day, guessWeekDay, guessYearDay;
    tm tmBuffer;
    char string[30];

    tmBuffer.tm_hour = 0;
    tmBuffer.tm_min = 0;
    tmBuffer.tm_sec = 0;
    tmBuffer.tm_isdst = -1;

    srand((unsigned)time(NULL));
    year = rand()%40+1970;
```

```
    month = rand()%12+1;
    day = rand()%30+1;
    tmBuffer.tm_year = year-1900;
    tmBuffer.tm_mon = month;
    tmBuffer.tm_mday = day;
    printf("What is the day of the week on %d/%d %d?\n", month, day,
year);
    printf("0 - Sunday, 1 - Monday, ...; your guess: ");
    scanf("%d", &guessWeekDay);
    mktime(&tmBuffer); // calculating week day and year day
    strftime(string, 30, "%A", &tmBuffer);
    if (tmBuffer.tm_wday == guessWeekDay) {
        printf("You are rigth %d/%d %d is really %s.\n", month, day, year,
string);
    } else {
        printf("You are wrong %d/%d %d is %s.\n", month, day, year,
string);
    }
    printf("\nWhat is the day of the year on the same date? ");
    scanf("%d", &guessYearDay);
    if (tmBuffer.tm_yday == guessYearDay) {
        printf("You are rigth %d/%d %d is the %d day of the year.\n",
month, day, year, tmBuffer.tm_yday);
    } else {
        printf("You are wrong %d/%d %d is the %d day of the year.\n",
month, day, year, tmBuffer.tm_yday);
    }
    return 0;
}
```

2.45.1.

```
#include <iostream>
#include <cmath>
using namespace std;

float heron(float a, float b, float c) {
    if (a<=0)
        throw a;
    if (b<=0)
        throw b;
    if (c<=0)
        throw c;
    if (a>=b+c || b>=a+c || c>=a+b)
        throw "No triangle can be constructed from the sides";
    float s=(a+b+c)/2;
    float result=sqrt(s*(s-a)*(s-b)*(s-c));
    return result;
}

int main() {
    cout << "The program will calculate the area of a triangle." << endl;
    float area;
    try {
        area=heron(3, 2.54, 9);
    } catch(float para) {
        cout << "Triangle side can not be 0 or less, it is " << para <<
endl;
    } catch(const char* msg) {
        cout << msg << endl;
```

```
    } catch(...) {
        cout << "Unknown error" << endl;
    }
    return 0;
}
```

2.46.1.

```
#include <iostream>
using namespace std;
class rightAngleTriangle {
private:
    float a; // befogo
    float b; // befogo
    float c; // atfogo
public:
    bool check() const;
    float area() const;
    float perimeter() const;
    void display() const;

    void setA(float para) {a=para;}
    void setB(float para) {b=para;}
    void setC(float para) {this->c=para;}
    float getA() const {return a;}
    float getB() const {return this->b;}
    float getC() const {return c;}
};
bool rightAngleTriangle::check() const {
    bool result=false;
    if (a*a+b*b==c*c)
        result=true;
    return result;
}
float rightAngleTriangle::area() const {
    float result=a*b/2;
    return result;
}
float rightAngleTriangle::perimeter() const {
    float result=a+b+c;
    return result;
}
void rightAngleTriangle::display() const {
    bool valid=check();
    cout << boolalpha;
    cout << "The triangle is valid: " << valid << endl;
    if (valid)
        cout << "The sides are: " << this->a << ", " << b << ", " <<
getC() << endl;
        // see the different references to the data members
}
int main() {
    rightAngleTriangle myObject;
    myObject.setA(3);
    myObject.setB(4);
    myObject.setC(5);
    myObject.display();
    if (myObject.check()) {
        cout << "Area: " << myObject.area() << endl;
        cout << "Perimeter: " << myObject.perimeter() << endl;
```

```
    }
    return 0;
}
```

2.46.2.

```
#include <iostream>
using namespace std;
class isoscelesTriangle {
private:
    float a; // befogo
    float b; // befogo
    float c; // atfogo
public:
    bool check() const;
    float area() const;
    float perimeter() const;
    void display() const;

    void setA(float para) {a=para;}
    void setB(float para) {b=para;}
    void setC(float para) {this->c=para;}
    float getA() const {return a;}
    float getB() const {return this->b;}
    float getC() const {return c;}
};
bool isoscelesTriangle::check() const {
    bool result=false;
    if (a==b || b==c || a==c)
        result=true;
    return result;
}
float isoscelesTriangle::area() const {
  float s=(a+b+c)/2;
    float result=s*(s-a)*(s-b);
    return result;
}
float isoscelesTriangle::perimeter() const {
    float result=a+b+c;
    return result;
}
void isoscelesTriangle::display() const {
    bool valid=check();
    cout << boolalpha;
    cout << "The triangle is valid: " << valid << endl;
    if (valid)
        cout << "The sides are: " << this->a << ", " << b << ", " <<
getC() << endl;
        // see the different references to the data members
}
int main() {
    isoscelesTriangle myObject;
    myObject.setA(3);
    myObject.setB(3);
    myObject.setC(5);
    myObject.display();
    if (myObject.check()) {
        cout << "Area: " << myObject.area() << endl;
        cout << "Perimeter: " << myObject.perimeter() << endl;
    }
```

```
    return 0;
}
```

2.46.3.

```
#include <iostream>
using namespace std;
class equilateralTriangle {
private:
    float a; // befogo
    float b; // befogo
    float c; // atfogo
public:
    bool check() const;
    float area() const;
    float perimeter() const;
    void display() const;

    void setA(float para) {a=para;}
    void setB(float para) {b=para;}
    void setC(float para) {this->c=para;}
    float getA() const {return a;}
    float getB() const {return this->b;}
    float getC() const {return c;}
};
bool equilateralTriangle::check() const {
    bool result=false;
    if (a==b && b==c)
        result=true;
    return result;
}
float equilateralTriangle::area() const {
  float s=(a+b+c)/2;
    float result=s*(s-a)*(s-b);
    return result;
}
float equilateralTriangle::perimeter() const {
    float result=a+b+c;
    return result;
}
void equilateralTriangle::display() const {
    bool valid=check();
    cout << boolalpha;
    cout << "The triangle is valid: " << valid << endl;
    if (valid)
        cout << "The sides are: " << this->a << ", " << b << ", " <<
getC() << endl;
        // see the different references to the data members
}
int main() {
    equilateralTriangle myObject;
    myObject.setA(3);
    myObject.setB(3);
    myObject.setC(5);
    myObject.display();
    if (myObject.check()) {
        cout << "Area: " << myObject.area() << endl;
        cout << "Perimeter: " << myObject.perimeter() << endl;
    }
    return 0;
```

```
}
```

2.47.1.

```
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;

class Counter {
    char fileName[500];
    ofstream file;
public:
    static int lineCount;
    void init(char* para);
    void close() {file.close();}
    void write(const char* line);
    static int getLineCount() {return lineCount;}
    static void resetLineCount() {lineCount=0;}
};

void Counter::init(char* para) {
    strcpy(fileName, para);
    file.open(fileName);
}

void Counter::write(const char* line) {
    file << line << endl;
    lineCount++;
}

int Counter::lineCount=0; //  class variable def.

int main() {
    Counter obj1, obj2;

    cout << "The number of lines have written till now:" <<
Counter::lineCount << endl;
    obj1.init("file1.txt");
    obj1.write("The first line");
    obj1.write("and the second");
    cout << "The number of lines have written till now:" <<
obj1.lineCount << endl;
    obj2.init("file2.txt");
    obj2.write("New line");
    cout << "The number of lines have written till now:" <<
obj2.lineCount << endl;
    Counter::resetLineCount();
    obj1.write("Strange line");
    cout << "The number of lines have written till now:" <<
obj2.lineCount << endl;
    obj1.close();
    obj2.close();
    return 0;
}
```

2.47.2.

```
#include <iostream>
#include <fstream>
#include <cstring>
```

```
using namespace std;

class Counter {
    char fileName[500];
    ofstream file;
public:
    static int lineCount;
    static int wordCount;
    static int charCount;
    void init(const char* para);
    void close() {file.close();}
    void write(const char* line);
};

void Counter::init(const char* para) {
    strcpy(fileName, para);
    file.open(fileName);
}

void Counter::write(const char* line) {
  int i=0;
  while (line[i]!='\0') {
    file << line[i];
    charCount++;
    if (line[i]==' ') { wordCount++; }
    i++;
  }
  if (line[0]!='\0') { wordCount++; }
    file << endl;
    lineCount++;
}

int Counter::lineCount=0; //  class variable def.
int Counter::wordCount=0; //  class variable def.
int Counter::charCount=0; //  class variable def.

int main() {
    Counter obj1, obj2;

    cout << "The number of lines have written till now:" <<
Counter::lineCount << endl;
    obj1.init("file1.txt");
    obj1.write("The first line");
    obj1.write("and the second");
    cout << "The number of lines have written till now:" <<
obj1.lineCount << endl;
    obj2.init("file2.txt");
    obj2.write("New line");
    cout << "The number of lines have written till now:" <<
obj2.lineCount << endl;
    obj2.lineCount=0;
    obj1.write("Strange line");
    cout << "The number of lines have written till now:" <<
obj2.lineCount << endl;
    obj1.close();
    obj2.close();
    return 0;
}
```

2.48.1.

```
#include <iostream>
#include <cstring>
using namespace std;

class Complex {
    float real, im;
    Complex multiply(const Complex&) const;
public:
    void setReal(float para) {real=para;}
    void setIm(float para) {im=para;}
    float getReal() const {return real;}
    float getIm() const {return im;}
    void display() const;
    void sum(const Complex&) const;
    void sub(const Complex&) const;
    void mul(const Complex&) const;
    void div(const Complex&) const;
};

void Complex::sum(const Complex& para) const {
    float resultReal, resultIm;
    resultReal=real+para.real;
    resultIm=im+para.im;
    cout << resultReal << "+" << resultIm << "i";
}

void Complex::sub(const Complex& para) const {
    float resultReal, resultIm;
    resultReal=real-para.real;
    resultIm=im-para.im;
    cout << resultReal << "+" << resultIm << "i";
}

void Complex::mul(const Complex& para) const {
    float resultReal, resultIm;
    resultReal=real*para.real - im*para.im;
    resultIm=im*para.real+real*para.im;
    cout << resultReal << "+" << resultIm << "i";
}

Complex Complex::multiply(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    resultReal=real*para.real - im*para.im;
    resultIm=im*para.real+real*para.im;
    result.real=resultReal;
    result.setIm(resultIm);
    return result;
}

void Complex::div(const Complex& para) const {
    float resultReal, resultIm;
    if (para.real==0 && para.im==0) {
        cout << "Error: divison by zero." << endl;
        return;
    }
    Complex recip;
```

```
    recip.real=para.real/(para.real*para.real+para.im*para.im);
    recip.setIm(-para.im/(para.real*para.real+para.im*para.im));
    Complex result=this->multiply(recip);
    resultReal=result.getReal();
    resultIm=result.im;
    cout << resultReal << "+" << resultIm << "i";
}

void Complex::display() const {
    cout << real << "+" << im << "i";
}

int main() {
    Complex a, b;

    a.setReal(1);
    a.setIm(2);
    b.setReal(3);
    b.setIm(4);
    a.display(); cout << " + "; b.display(); cout << "="; a.sum(b); cout
<< endl;
    a.display(); cout << " - "; b.display(); cout << "="; a.sub(b); cout
<< endl;
    a.display(); cout << " * "; b.display(); cout << "="; a.mul(b); cout
<< endl;
    a.display(); cout << " / "; b.display(); cout << "="; a.div(b);  cout
<< endl;
    return 0;
}
```

2.48.2.

```
#include <iostream>
#include <cstring>
using namespace std;

class Complex {
    float real, im;
    Complex multiply(const Complex&) const;
public:
    void setReal(float para) {real=para;}
    void setIm(float para) {im=para;}
    float getReal() const {return real;}
    float getIm() const {return im;}
    void display() const;
    Complex sum(const Complex&) const;
    Complex sub(const Complex&) const;
    Complex mul(const Complex&) const;
    Complex div(const Complex&) const;
    friend ostream& operator <<(ostream &os,const Complex &obj);
};

ostream& operator <<(ostream &os,const Complex &obj) {
  os<< "(" << obj.real << " + " << obj.im << "i)";
  return os;
}

Complex Complex::sum(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
```

```
    resultReal=real+para.real;
    resultIm=im+para.im;
    result.real=resultReal;
    result.setIm(resultIm);
    return result;
}

Complex Complex::sub(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    resultReal=real-para.real;
    resultIm=im-para.im;
    result.real=resultReal;
    result.setIm(resultIm);
    return result;
}

Complex Complex::mul(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    resultReal=real*para.real - im*para.im;
    resultIm=im*para.real+real*para.im;
    result.real=resultReal;
    result.setIm(resultIm);
    return result;
}

Complex Complex::multiply(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    resultReal=real*para.real - im*para.im;
    resultIm=im*para.real+real*para.im;
    result.real=resultReal;
    result.setIm(resultIm);
    return result;
}

Complex Complex::div(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    if (para.real==0 && para.im==0) {
        cout << "Error: divison by zero." << endl;
        return result;
    }
    Complex recip;
    recip.real=para.real/(para.real*para.real+para.im*para.im);
    recip.setIm(-para.im/(para.real*para.real+para.im*para.im));
    result=this->multiply(recip);
    resultReal=result.getReal();
    resultIm=result.im;
    return result;
}

void Complex::display() const {
    cout << real << " + " << im << "i";
}

int main() {
    Complex a, b;
```

```
    a.setReal(1);
    a.setIm(2);
    b.setReal(3);
    b.setIm(4);
    cout << a << " + " << b << "=" << a.sum(b) << endl;
    cout << a << " - " << b << "=" << a.sub(b) << endl;
    cout << a << " * " << b << "=" << a.mul(b) << endl;
    cout << a << " / " << b << "=" << a.div(b) << endl;
    return 0;
}
```

2.48.3.

```
#include <iostream>
#include <cstring>
using namespace std;

class Complex {
    float real, im;
    Complex multiply(const Complex&) const;
public:
    void setReal(float para) {real=para;}
    void setIm(float para) {im=para;}
    float getReal() const {return real;}
    float getIm() const {return im;}
    void display() const;
    Complex sum(const Complex&) const;
    Complex sub(const Complex&) const;
    Complex mul(const Complex&) const;
    Complex div(const Complex&) const;
    friend ostream& operator <<(ostream &os,const Complex &obj);
    const Complex operator+(const Complex &other) const {
     Complex result = sum(other);
     return result;
  }
    const Complex operator-(const Complex &other) const {
     Complex result = sub(other);
     return result;
  }
    const Complex operator*(const Complex &other) const {
     Complex result = mul(other);
     return result;
  }
    const Complex operator/(const Complex &other) const {
     Complex result = div(other);
     return result;
  }
};

ostream& operator <<(ostream &os,const Complex &obj) {
  os<< "(" << obj.real << " + " << obj.im << "i) ";
  return os;
}

Complex Complex::sum(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    resultReal=real+para.real;
    resultIm=im+para.im;
```

```
    result.real=resultReal;
    result.setIm(resultIm);
    return result;
}

Complex Complex::sub(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    resultReal=real-para.real;
    resultIm=im-para.im;
    result.real=resultReal;
    result.setIm(resultIm);
    return result;
}

Complex Complex::mul(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    resultReal=real*para.real - im*para.im;
    resultIm=im*para.real+real*para.im;
    result.real=resultReal;
    result.setIm(resultIm);
    return result;
}

Complex Complex::multiply(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    resultReal=real*para.real - im*para.im;
    resultIm=im*para.real+real*para.im;
    result.real=resultReal;
    result.setIm(resultIm);
    return result;
}

Complex Complex::div(const Complex& para) const {
    float resultReal, resultIm;
    Complex result;
    if (para.real==0 && para.im==0) {
        cout << "Error: divison by zero." << endl;
        return result;
    }
    Complex recip;
    recip.real=para.real/(para.real*para.real+para.im*para.im);
    recip.setIm(-para.im/(para.real*para.real+para.im*para.im));
    result=this->multiply(recip);
    resultReal=result.getReal();
    resultIm=result.im;
    return result;
}

void Complex::display() const {
    cout << real << " + " << im << "i";
}

int main() {
    Complex a, b;

    a.setReal(1);
```

```
    a.setIm(2);
    b.setReal(3);
    b.setIm(4);
    cout << a << " + " << b << "=" << a + b << endl;
    cout << a << " - " << b << "=" << a - b << endl;
    cout << a << " * " << b << "=" << a * b << endl;
    cout << a << " / " << b << "=" << a / b << endl;
    return 0;
}
```

2.49.1.

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

#define NEV_HOSZ 11
#define INPUT_FILE "in.txt"

struct TSubscriber
{
   char FirstName[NEV_HOSZ];
   char LastName[NEV_HOSZ];
   int InRushH, OutRushH;
   int TotalMin, Fee;
};

void read(FILE * fd, int num, struct TSubscriber * Subscribers)
{
     int i;

     for (i = 0; i < num; i++)
     {
         fscanf(fd, "%s %s", Subscribers[i].FirstName,
Subscribers[i].LastName);
         fscanf(fd, "%d %d", &Subscribers[i].InRushH,
&Subscribers[i].OutRushH);
     }
}

void calc(int num, struct TSubscriber * A, int csucs, int kedv)
{
  int i;
  int MaxMin = 0;
  int MaxFee = 0;
  for (i = 0; i < num; i++)
  {
      A[i].Fee = A[i].InRushH * csucs + A[i].OutRushH * kedv;
      A[i].TotalMin = A[i].InRushH + A[i].OutRushH;
      printf("\t\t\t %5d Ft %5d minute\r%s %s :\n", A[i].Fee,
A[i].TotalMin,
      A[i].FirstName, A[i].LastName);
      if (MaxMin < A[i].TotalMin)
         MaxMin = A[i].TotalMin;
      if (MaxFee < A[i].Fee)
         MaxFee = A[i].Fee;
  }
  printf("\nHighest fee: %d Ft\n", MaxFee);
  for (i=0; i<num; i++)
  {
      if (A[i].Fee == MaxFee)
```

```
        printf("\t%s %s\n", A[i].FirstName, A[i].LastName);
  }
  printf("\nLongest speaking: %d Ft\n", MaxMin);
  for (i=0; i<num; i++)
  {
      if (A[i].TotalMin == MaxMin)
         printf("\t%s %s\n",A[i].FirstName, A[i].LastName);
  }
}

int main()
{
  struct TSubscriber * Subscribers = NULL;
  int inrush, outrush, num;
  FILE * fd = fopen(INPUT_FILE, "r");
  if (fd == NULL)
  {
     perror("Hiba");
     return 0;
  }
  fscanf(fd, "%d", &num);
  fscanf(fd, "%d %d", &inrush,&outrush);
  Subscribers = (struct TSubscriber *)malloc(num * sizeof(struct
TSubscriber));

  read(fd, num, Subscribers );
  fclose(fd);

  calc(num, Subscribers, inrush, outrush);
  free(Subscribers);
  Subscribers = NULL;

  return 0;
}
```

2.50.1.

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

#define INPUT_FILE "sets.txt"
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) > (b) ? (b) : (a))

void ReadSet(FILE * fd, int * size, int ** set)
{
  int i;
  fscanf(fd, "%d", size);
  *set = (int *)malloc(sizeof(int) * (*size) );
  for (i = 0; i < *size; i++)
    fscanf(fd, "%d", *set + i);
  printf("\n");
}

void PrintSet(char * name, int size, int * set)
{
  int i;
  printf("%s = ", name);
  printf("{");
  for (i = 0; i < size - 1; i++)
```

```
    printf("%d, ", set[i]);
  if (size > 0)
       printf("%d", set[size - 1]);
  printf("}\n");
}

int Element(int e, int size, int * S)
{
  int i = 0;
  while ((i < size) && (S[i] != e))
    i++;
  return i < size;
}

void Intersection(int SizeA, int SizeB, int * SizeI, int * SA, int * SB,
int ** C)
{
   int size = 0;
   int i;
   *SizeI = 0;

   int * A = SizeA > SizeB ? SA : SB;
   int * B = SizeA > SizeB ? SB : SA;

   for (i = 0; i < max(SizeA, SizeB); i++)
   {

      if (Element(A[i], max(SizeA, SizeB), A) && Element(A[i], min(SizeA,
SizeB), B) )
        (*SizeI)++;
   }

   *C = (int *)malloc(sizeof(int) * (*SizeI) );
   for (i = 0; i < max(SizeA, SizeB); i++)
   {

      if (Element(A[i], max(SizeA, SizeB), A) && Element(A[i], min(SizeA,
SizeB), B) )
        (*C)[size++] = A[i];
   }
}

int IsSubset(int SizeA, int SizeS, int * A, int * S)
{
  int i;
  if (SizeS > SizeA)
    return 0;
  for (i = 0; i < SizeS; i++)
    if (!Element(S[i], SizeA, A))
      return 0;
  return 1;
}

int main(int argc, char *argv[])
{
  int SetASize, SetBSize, SetISize;
  int * SetA;
  int * SetB;
  int * SetI;
```

```
  int num;
  FILE * fd = fopen(argc > 1 ? argv[1] : INPUT_FILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  ReadSet(fd, &SetASize, &SetA);
  ReadSet(fd, &SetBSize, &SetB);
  fclose(fd);
  Intersection(SetASize, SetBSize, &SetISize, SetA, SetB, &SetI);

  printf("Type a number: ");
  scanf("%d", &num);
  if (Element(num, SetASize, SetA))
    printf("%d is element of A\n", num);
  else
    printf("%d is not element of A\n", num);

  if (Element(num, SetBSize, SetB))
    printf("%d is element of B\n", num);
  else
    printf("%d is not element of B\n", num);

  if (Element(num, SetISize, SetI))
    printf("%d is element of intersection of A and B\n", num);
  else
    printf("%d is not element of intersection of A and B\n", num);

  PrintSet("A", SetASize, SetA);
  PrintSet("B", SetBSize, SetB);
  PrintSet("Intersection of A and B", SetISize, SetI);
  if (IsSubset(SetASize, SetBSize, SetA, SetB))
    printf("B is subset of A\n");
   else
    printf("B is not subset of A\n");
  if (IsSubset(SetBSize, SetASize, SetB, SetA))
    printf("A is subset of B\n");
   else
    printf("A is not subset of B\n");

  free(SetA); SetA = NULL;
  free(SetB); SetB = NULL;
  free(SetI); SetI = NULL;
  return 0;
}
```

2.51.1.

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

#define INPUT_FILE "sets.txt"
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) > (b) ? (b) : (a))

void ReadSet(FILE * fd, int * size, int ** set)
{
  int i;
  fscanf(fd, "%d", size);
```

```
  *set = (int *)malloc(sizeof(int) * (*size) );
  for (i = 0; i < *size; i++)
    fscanf(fd, "%d", *set + i);
  printf("\n");
}

void PrintSet(char * name, int size, int * set)
{
  int i;
  printf("%s = ", name);
  printf("{");
  for (i = 0; i < size - 1; i++)
    printf("%d, ", set[i]);
  if (size > 0)
       printf("%d", set[size - 1]);
  printf("}\n");
}

int Element(int e, int size, int * S)
{
  int i = 0;
  while ((i < size) && (S[i] != e))
    i++;
  return i < size;
}

void Union(int SizeA, int SizeB, int * SizeU, int * SA, int * SB, int **
C)
{
  int size = max(SizeA, SizeB);
  int i;
  *SizeU = 0;
   int * A = SizeA > SizeB ? SA : SB;
   int * B = SizeA > SizeB ? SB : SA;

  for (i = 0; i < min(SizeA, SizeB); i++)
  {
      if ( !Element(B[i], max(SizeA, SizeB), A) )
        size++;
  }
  *C = (int *)malloc(sizeof(int) * size );
  *SizeU = size;
  for (i = 0; i < max(SizeA, SizeB); i++)
    (*C)[i] = A[i];

  size = max(SizeA, SizeB);
  for (i = 0; i < min(SizeA, SizeB); i++)
  {
      if ( !Element(B[i], max(SizeA, SizeB), A) )
        (*C)[size++] = B[i];
  }
}

int IsSubset(int SizeA, int SizeS, int * A, int * S)
{
  int i;
  if (SizeS > SizeA)
    return 0;
  for (i = 0; i < SizeS; i++)
```

```
    if (!Element(S[i], SizeA, A))
      return 0;
  return 1;
}

int main(int argc, char *argv[])
{
  int SetASize, SetBSize, SetUSize;
  int * SetA;
  int * SetB;
  int * SetU;
  int num;
  FILE * fd = fopen(argc > 1 ? argv[1] : INPUT_FILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  ReadSet(fd, &SetASize, &SetA);
  ReadSet(fd, &SetBSize, &SetB);
  fclose(fd);
  Union(SetASize, SetBSize, &SetUSize, SetA, SetB, &SetU);


  printf("Type a number: ");
  scanf("%d", &num);
  if (Element(num, SetASize, SetA))
    printf("%d is element of A\n", num);
  else
    printf("%d is not element of A\n", num);

  if (Element(num, SetBSize, SetB))
    printf("%d is element of B\n", num);
  else
    printf("%d is not element of B\n", num);

  if (Element(num, SetUSize, SetU))
    printf("%d is element of union of A and B\n", num);
  else
    printf("%d is not element of union of A and B\n", num);


  PrintSet("A", SetASize, SetA);
  PrintSet("B", SetBSize, SetB);
  PrintSet("Union of A and B", SetUSize, SetU);
  if (IsSubset(SetASize, SetBSize, SetA, SetB))
    printf("B is subset of A\n");
   else
    printf("B is not subset of A\n");
  if (IsSubset(SetBSize, SetASize, SetB, SetA))
    printf("A is subset of B\n");
   else
    printf("A is not subset of B\n");

  free(SetA); SetA = NULL;
  free(SetB); SetB = NULL;
  free(SetU); SetU = NULL;
  return 0;
}
```

2.52.1.

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

#define INPUT_FILE "sets.txt"
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) > (b) ? (b) : (a))

void ReadSet(FILE * fd, int * size, int ** set)
{
  int i;
  fscanf(fd, "%d", size);
  *set = (int *)malloc(sizeof(int) * (*size) );
  for (i = 0; i < *size; i++)
    fscanf(fd, "%d", *set + i);
  printf("\n");
}

void PrintSet(char * name, int size, int * set)
{
  int i;
  printf("%s = ", name);
  printf("{");
    for (i = 0; i < size - 1; i++)
    printf("%d, ", set[i]);
    if (size > 0)
        printf("%d", set[size - 1]);
  printf("}\n");
}

int Element(int e, int size, int * S)
{
  int i = 0;
  while ((i < size) && (S[i] != e))
    i++;
  return i < size;
}

// A \ B
void SetDifference(int SizeA, int SizeB, int * SizeD, int * SA, int * SB,
int ** C)
{
  int size = 0;
  int i;
  for (i = 0; i < SizeA; i++)
  {
    if (!Element(SA[i], SizeB, SB))
      size++;
  }
  *C = (int *)malloc(sizeof(int) * size);
  *SizeD = size;
  size = 0;
  for (i = 0; i < SizeA; i++)
  {
    if (!Element(SA[i], SizeB, SB))
      (*C)[size++] = SA[i];
  }
}
```

```
int IsSubset(int SizeA, int SizeS, int * A, int * S)
{
  int i;
  if (SizeS > SizeA)
    return 0;
  for (i = 0; i < SizeS; i++)
    if (!Element(S[i], SizeA, A))
      return 0;
  return 1;
}

int main(int argc, char *argv[])
{
  int SetASize, SetBSize, SetD1Size, SetD2Size;
  int * SetA;
  int * SetB;
  int * SetD1;
  int * SetD2;
  int num;
  FILE * fd = fopen(argc > 1 ? argv[1] : INPUT_FILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  ReadSet(fd, &SetASize, &SetA);
  ReadSet(fd, &SetBSize, &SetB);
  fclose(fd);

  SetDifference(SetASize, SetBSize, &SetD1Size, SetA, SetB, &SetD1);
  SetDifference(SetBSize, SetASize, &SetD2Size, SetB, SetA, &SetD2);

  printf("Type a number: ");
  scanf("%d", &num);
  if (Element(num, SetASize, SetA))
    printf("%d is element of A\n", num);
  else
    printf("%d is not element of A\n", num);

  if (Element(num, SetBSize, SetB))
    printf("%d is element of B\n", num);
  else
    printf("%d is not element of B\n", num);

  if (Element(num, SetD1Size, SetD1))
    printf("%d is element of A \\ B\n", num);
  else
    printf("%d is not element of A \\ B\n", num);

  if (Element(num, SetD2Size, SetD2))
    printf("%d is element of B \\ A\n", num);
  else
    printf("%d is not element of B \\ A\n", num);

  PrintSet("A", SetASize, SetA);
  PrintSet("B", SetBSize, SetB);
  PrintSet("Set difference A\\B", SetD1Size, SetD1);
  PrintSet("Set difference B\\A", SetD2Size, SetD2);
```

```
  if (IsSubset(SetASize, SetBSize, SetA, SetB))
    printf("B is subset of A\n");
    else
    printf("B is not subset of A\n");
  if (IsSubset(SetBSize, SetASize, SetB, SetA))
    printf("A is subset of B\n");
    else
    printf("A is not subset of B\n");

  free(SetA); SetA = NULL;
  free(SetB); SetB = NULL;
  free(SetD1); SetD1 = NULL;
  free(SetD2); SetD2 = NULL;
  return 0;
}
```

2.53.1.

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

typedef unsigned long int ULint;

ULint BinomialCoefficient(int n, int k)
{
  ULint bc = n;
  int i;
  if (k == 0)
    return 1;
  for (i = n - 1 ; i > (n - k); i--)
  {
    bc *= i;
//    printf("%d %ld\n", i, bc);
  }
  for (i = 2; i <= k; i++)
    bc /= i;
  return bc;
}

void PrintPow(char ch, int p)
{
  if (p == 0)
    return;
  if (p == 1)
    printf("%c", ch);
  else
    printf("%c^%d", ch, p);
}

void PrintBinom(int a)
{
  int i;
  int b;
  printf("( a + b )^%d = ", a);
  for (i = 0; i <= a; i++)
  {
    b = BinomialCoefficient(a, i);
    if (b > 1)
      printf("%d", b);
    PrintPow('b', i);
```

```
    PrintPow('a', a - i);
    if (i < a)
      printf(" + ");
  }
  printf("\n");
}

int main(int argc, char * argv[] )
{
  int a;
  if (argc > 1)
    a = atoi(argv[1]);
  else
  {
    printf(„Degree: „);
    scanf("%d", &a);
  }
  printf("\n");
  PrintBinom(a);
  printf("\n");
  return 0;
}
```

2.54.1.

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

#define MIN_TIP 1
#define EASY_MAX 5
#define MEDIUM_MAX 10
#define HARD_MAX 15
#define START_MONEY 1000

int GetStake()
{
  int stake;
  do
  {
    printf("Take your stake: ");
    scanf("%d", &stake);
    if (stake <= 0)
      printf("The stake have to higher than zero!\n");
  } while (stake <= 0);
  return stake;
}

int GetTip(int min, int max)
{
  int tip;
  do
  {
    printf("What's your tip (%d..%d)? ", min, max);
    scanf("%d", &tip);
    if ((tip > max) || (tip < min))
      printf("The tip have to be between %d and %d!\n", min, max);
  } while ((tip > max) || (tip < min));
  return tip;
}
```

```
int GetMax()
{
  int d;
  do
  {
    printf("Choose the difficulty!\n\n");
    printf("\tEasy:.....1\n");
    printf("\tMedium:.....2\n");
    printf("\tHard:.....3\n\n");
    scanf("%d", &d);
    if ((d > 3) || (d < 1))
      printf("Wrong number!\n");
  } while ((d > 3) || (d < 1));
  switch (d)
  {
    case 1:
      return EASY_MAX;
    case 2:
      return MEDIUM_MAX;
    case 3:
      return HARD_MAX;
  }
  return 0;
}

int Random(int min, int max)
{
  return (rand() % (max - min)) + min;
}

int main()
{
  int stake, tip;
  int max;
  int num;
  int Money = START_MONEY;
  char s[5];
  srand(time(NULL));
  max = GetMax();
  printf("You have %d HUF now.\n", Money);
  do
  {
    printf("\n\n");
    stake = GetStake();
    num = Random(MIN_TIP, max);
    tip = GetTip(MIN_TIP, max);

    if (tip == num)
    {
      Money += stake;
      printf("You've hit the number!\nYou have %d HUF!\n", Money);
    } else
    {
      printf("The number was %d\n", num);
      Money -= stake;
      printf("You have %d HUF!\n", Money);
    }
    if (Money > 0)
```

```
    {
      printf("Try again? (y/n)\n");
      scanf("%s", s);
    }
  } while (((s[0] == 'y') || (s[0] == 'Y')) && (Money > 0));
  if (Money > 0)
    printf("Congratulations!\nYou get %d HUF!\n", Money);
  else
    printf("I'm sorry, you lost\n");
  srand(time(NULL));
  return 0;
}
```

2.55.1.

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

#define DEFAULT_INPUTFILE "date.txt"
#define DOT '.'
#define MAX_LENGTH 10

struct TDate
{
   char Year[MAX_LENGTH];
   char Month[MAX_LENGTH];
   char Day[MAX_LENGTH];
};

void InitDate(struct TDate * Date)
{
   Date->Year[0] = Date->Month[0] = Date->Day[0] = 0;
}

void ReadDate(FILE * fd, struct TDate * Date)
{
     fscanf(fd, "%s", Date->Year);
     fscanf(fd, "%s", Date->Month);
     fscanf(fd, "%s", Date->Day);
}

void PrintDate(struct TDate * Date)
{
   printf("%s %s %s\n", Date->Year, Date->Month, Date->Day);
}

int CheckNumAndDots(char * S, char * A)
{
     int i;
     int len = strlen(S);
     int errnum = 0;
     /* Check numbers: */
     for (i = 0; i < len - 1; i++)
     {
         if ((S[i] > '9') || (S[i] < '0'))
         {
            errnum++;
            printf("Error: the %s cannot contain %c character!\n", A,
S[i]);
```

```
         }
      }
      /* Check dot: */
      if (S[len - 1] != DOT)
      {
         errnum++;
         printf("Error: A \"%c\" have to follow the number, not
\"%c\"!\n", DOT, S[len - 1]);
      }
      return errnum;
}

int CheckNums(struct TDate * Date)
{
   int errnum = 0;
   int month = atoi(Date->Month);
   int day = atoi(Date->Day);
   if ((month > 12) || (month < 1))
   {
      errnum++;
      printf("Error: There are only 12 months!\n");
   }
   if ((day > 31) || (day < 1))
   {
      errnum++;
      printf("Error: There are atmost 31, at least 1 days in a
month!\n");
   }
   return errnum;
}

void CheckDate(struct TDate * Date)
{
      int errnum = CheckNumAndDots(Date->Year, "Year");
      errnum += CheckNumAndDots(Date->Month, "Month");
      errnum += CheckNumAndDots(Date->Day, "Day");
      if (errnum == 0)
         errnum += CheckNums(Date);
      printf("*****************************\nI found %d errors\n",
errnum);
}

int main(int argc, char *argv[])
{
  struct TDate Date;
  InitDate(&Date);
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == 0)
  {
     perror("Error");
     return 0;
  }
  ReadDate(fd, &Date);
  PrintDate(&Date);
  CheckDate(&Date);
  fclose(fd);
  return 0;
}
```

2.55.2.

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

#define DEFAULT_INPUTFILE "date.txt"
#define YEAR_MIN 1000
#define YEAR_MAX 3000
#define MONTH_MAX 12
#define DAY_MAX 31

const char * Months_Strings[12] = { "January", "February", "March",
"April", "May", "June",
       "July", "August", "September", "October", "November", "December"
};

enum Months{ jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov,
dec};

struct TDate
{
   int Year;
   int Month;
   int Day;
};

void ReadDate(FILE * fd, struct TDate * Date)
{
     fscanf(fd, "%d.", &Date->Year);
     fscanf(fd, "%d.", &Date->Month);
     fscanf(fd, "%d.", &Date->Day);
}

void PrintDate(struct TDate * Date)
{
     printf("Year: %d\nMonth: %s\nDay: %d\n", Date->Year,
                    ((Date->Month >= 1) && (Date->Month <= 12)) ?
Months_Strings[Date->Month - 1]
                                       : "Bad month", Date->Day);
}

int CheckYear(int Y)
{
    int errnum = 0;
    if (Y > YEAR_MAX)
    {
       errnum++;
       printf("Error: %d is too high for the year!\n", Y);
    }
    if (Y < YEAR_MIN)
    {
       errnum++;
       printf("Error: %d is too low for the year!\n", Y);
    }
    return errnum;
}

int CheckMonth(int M)
{
```

```
    int errnum = 0;
    if (M > MONTH_MAX)
    {
       errnum++;
       printf("Error: %d is too high for the month!\n", M);
    }
    if (M < 1)
    {
       errnum++;
       printf("Error: %d is too low for the month!\n", M);
    }
    return errnum;
}

int LeapYear(int y)
{
  if (y % 4 == 0)
  {
    if (y % 100 == 0)
    {
      if (y % 400 == 0)
        return 1;
      else
        return 0;
    } else return 1;
  }
  return 0;
}

int CheckDay(struct TDate * Date)
{
    int errnum = 0;
    switch (Date->Month)
    {
           case jan:
           case mar:
           case may:
           case jul:
           case aug:
           case oct:
           case dec:
                if (Date->Day > 31)
                {
                   errnum++;
                   printf("Error: In %s there are 31 days!\n",
Months_Strings[ Date-> Month - 1]);
                }
                break;
           case apr:
           case jun:
           case sep:
           case nov:
                if (Date->Day > 30)
                {
                   errnum++;
                   printf("Error: In %s there are 30 days!\n",
Months_Strings[ Date-> Month - 1]);
                }
                break;
```

```
            case feb:
                 /* Leap year? */
                 if (LeapYear(Date->Year))
                 {
                    if (Date->Day > 29)
                    {
                       errnum++;
                       printf("Error: In %dth %s there are 29 days!\n",
Date->Year,
                        Months_Strings[ Date-> Month - 1 ]);
                    }
                 } else
                 {
                    if (Date->Day > 28)
                    {
                       errnum++;
                       printf("Error: In %dth %s there are 28 days!\n",
Date->Year,
                        Months_Strings[ Date-> Month - 1 ]);
                    }
                 }
                 break;
    }
    return errnum;
}


void CheckDate(struct TDate * Date)
{
     int errnum = CheckYear(Date->Year);
     errnum += CheckMonth(Date->Month);
     if (!errnum)
        errnum = CheckDay(Date);
     printf("*****************************\nI found %d errors\n",
errnum);
}

int main(int argc, char *argv[])
{
  struct TDate Date;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == 0)
  {
     perror("Error");
     return 0;
  }
  ReadDate(fd, &Date);
  PrintDate(&Date);
  CheckDate(&Date);
  fclose(fd);
  return 0;
}
```

2.56.1.

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

#define DEFAULT_ROWNUM 10
#define SZOKOZS 4
```

```
int BinomialCoefficient(int n, int k)
{
  int bc = n;
  int i;
  if (k == 0)
    return 1;
  for (i = n - 1 ; i > (n - k); i--)
  {
    bc *= i;
  }
  for (i = 2; i <= k; i++)
    bc /= i;
  return bc;
}

int GetWidth(int n)
{
  return n * 3 + (n - 1) * SZOKOZS;
}

void PascalsTriangle(int rows)
{
  int i, sp, r, j;
  int width = GetWidth(rows);
  for (r = 0; r < rows; r++)
  {
    for (sp = 0; sp < (width / 2) - (GetWidth(r) / 2); sp++)
      printf("");
    for (i = 0; i <= r; i++)
    {
      printf("%3d", BinomialCoefficient(r, i));
      for (j = 0; j < SZOKOZS; j++)
        printf("");
    }

    printf("\n");
  }
}

int main(int argc, char * argv[])
{
  int rows = argc > 1 ? atoi(argv[1]) : DEFAULT_ROWNUM;
  PascalsTriangle(rows);
  return 0;
}
```

2.57.1.

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

#define DEFAULT_INPUTFILE "tipp1.txt"

void GetNumbers(int * A)
{
  int i, j, num;
  for (i = 0; i < 5; i++)
  {
```

```
    do
    {
      num = rand() % 90 + 1;
      j = 0;
      while ((j < i) && (A[j] != num))
        j++;
    } while (j < i);
    A[i] = num;
  }
}

void PrintNumbers(int * A)
{
  int i;
  for (i = 0; i < 4; i++)
    printf("%d, ", A[i]);
  printf("%d\n", A[4]);
}

void Check(FILE * fd, int * A, int * H)
{
  int i, j, k, c, num, a;
  fscanf(fd, "%d", &c);
  for (k = 0; k < c; k++)
  {
    a = 0;
    for (i = 0; i < 5; i++)
    {
      fscanf(fd, "%d", &num);
      j = 0;
      while ((j < 5) && (A[j] != num))
        j++;
      if (j < 5)
        a++;
    }
    if (a > 0)
      H[a - 1]++;
  }
}

void InitHits(int * A)
{
  int i;
  for (i = 0; i < 5; i++)
    A[i] = 0;
}

void PrintHits(int * A)
{
  int i;
  for (i = 0; i < 5; i++)
  {
    printf("There ");
    if (A[i] > 1)
      printf("are ");
    else
      printf("is ");
    printf("%d %d hit", A[i], i + 1);
    if (A[i] > 1)
```

```
      printf("s\n");
    else
      printf("\n");
  }
}

int main(int argc, char * argv[])
{
  int Numbers[5];
  int Hits[5];
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, „r”);
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  srand(time(NULL));
  InitHits(Hits);
  GetNumbers(Numbers);
  printf("Generated numbers: ");
  PrintNumbers(Numbers);
  Check(fd, Numbers, Hits);
  PrintHits(Hits);
  fclose(fd);
  return 0;
}
```

2.58.1.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define DEFAULT_RINGNUM 3
#define TRUE 1
#define FALSE 0
#define NAMELEN 12

struct TTower
{
  int MaxHeight;
  int Height;
  int * Disks;
  char Name[NAMELEN];
};

void InitTower(struct TTower * T, int height, int full, char * name)
{
  int i;
  T->Disks = (int*)malloc(sizeof(int) * height);
  T->MaxHeight = height;
  strcpy(T->Name, name);
  if (full)
  {
     for (i = height; i >= 1; i--)
         T->Disks[height - i] = i;
     T->Height = height;
  } else
  {
     for (i = 0; i < height; i++)
         T->Disks[i] = 0;
```

```
     T->Height = 0;
  }
}

void FreeTower(struct TTower * T)
{
     free(T->Disks);
     T->Disks = NULL;
     T->Height = T->MaxHeight = 0;
}

void PrintTower(struct TTower * T)
{
  int i;
  printf("%s: ", T->Name);
  for (i = 0; i < T->Height - 1; i++)
      printf("%d, ", T->Disks[i]);
  printf("%d\n", T->Disks[i]);
}

int GetIndex(char * S)
{
    if (strcmp(S, "src") == 0)
       return 0;
    if (strcmp(S, "aux") == 0)
       return 1;
    if (strcmp(S, "dst") == 0)
       return 2;
    return -1;
}

int IsCorrect(int f, int t, struct TTower * Towers)
{
    if ((f == -1) || (t == -1))
       return FALSE;
    if (f == t)
       return FALSE;
    if (Towers[f].Height == 0)
       return FALSE;
    if (Towers[f].Disks[Towers[f].Height - 1] >
Towers[t].Disks[Towers[t].Height - 1])
       return FALSE;
    return TRUE;
}

int isComplete(struct TTower * T)
{
  int i;
  for (i = T->MaxHeight; i >= 1; i--)
      if (T->Disks[ T->MaxHeight - i] != i)
         return FALSE;
  return TRUE;
}

int main(int argc, char * argv[])
{
    char From[5];
    char To[5];
    int f, t;
```

```
    int giveup;
    struct TTower Towers[3];
    int MaxH = argc > 1 ? atoi(argv[1]) : DEFAULT_RINGNUM;
    InitTower(Towers, MaxH, TRUE, "Source");
    InitTower(Towers + 1, MaxH, FALSE, "Auxiliary");
    InitTower(Towers + 2, MaxH, FALSE, "Destination");
    do
    {
      printf("\n\n************************************\n\n");
      PrintTower(Towers);
      PrintTower(Towers + 1);
      PrintTower(Towers + 2);
      printf("From: ");
      scanf("%s", From);
      if (From[0] != '0')
      {
         printf("To: ");
         scanf("%s", To);
      }
      giveup = ((From[0] == '0') || (To[0] == '0'));
      if (!giveup)
      {
         f = GetIndex(From);
         t = GetIndex(To);
         if (IsCorrect(f, t, Towers))
         {
            Towers[t].Disks[Towers[t].Height++] =
Towers[f].Disks[Towers[f].Height - 1];
            Towers[f].Disks[--Towers[f].Height] = 0;
         } else printf("Incorrect step!\n");
      }
    } while ((!giveup) && !isComplete(Towers + 2));
    if (!giveup)
       printf("The disks are on their's place!\n");
    else
        printf("The disks are not on the destination!\n");

    printf("\n\n************************************\n\n");
    PrintTower(Towers);
    PrintTower(Towers + 1);
    PrintTower(Towers + 2);
    FreeTower(Towers);
    FreeTower(Towers + 1);
    FreeTower(Towers + 2);
    return 0;
}
```

2.59.1.

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

#define DEFAULT_INPUTFILE "nums.txt"

int Eukl(int a, int b)
{
  int m = b;
  while (m != 0)
  {
    b = m;
```

```
    m = a % b;
    a = b;
  }
  return b;
}

void ReadAndAdd(FILE * fd)
{
  int count, i;
  int num1, num2;
  int den = 0, numer = 0;
  int tmp1, tmp2;
  int d;
  fscanf(fd, "%d", &count);
  for (i = 0; i < count; i++)
  {
    fscanf(fd, "%d %d", &num1, &num2);
    if (den != 0)
    {
      tmp2 = den * num2;
      tmp1 = num2 * numer + den * num1;
      d = Eukl(tmp1, tmp2);
      if (d != 1)
      {
        printf("%4d     %4d      %4d       %4d\n", numer, num1, tmp1,
tmp1 / d);
        printf("------ + ------ =  ------ =  ------\n");
        printf("%4d     %4d      %4d       %4d\n\n", den, num2, tmp2,
tmp2 / d);
      } else
      {
        printf("%4d     %4d      %4d\n", numer, num1, tmp1);
        printf("------ + ------ =  ------\n");
        printf("%4d     %4d      %4d\n\n", den, num2, tmp2);
      }
      numer = tmp1 / d;
      den = tmp2 / d;
    }else
    {
      den = num2;
      numer = num1;
    }

  }
}

int main(int argc, char * argv[])
{
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, „r”);
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  ReadAndAdd(fd);
  fclose(fd);
  return 0;
}
```

2.60.1-2.60.4.

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

int osztokOsszege(int szam) {
    if (szam == 1) {
        return 0;
    }
    int oszto = 2;
    int osszeg = 1;
    while (oszto * oszto < szam) {
        if (szam % oszto == 0) {
            osszeg += oszto;
            osszeg += szam / oszto;
        }
        oszto++;
    }
    return osszeg;
}

#define N1 10000
#define N2 100
#define N3 1000

int main() {
    int i;
    int tokeletes = 0;
    int hianyos = 0;
    int bovelkedo = 0;
    int bovelkedo2 = 0;
    for (i = 1; i <= N1; i++) {
        if (osztokOsszege(i) == i) {
            tokeletes++;
        }
    }

    for (i = 1; i <= N2; i++) {
        if (osztokOsszege(i) < i) {
            hianyos++;
        }
    }

    for (i = 1; i <= N3; i++) {
        if (osztokOsszege(i) > i) {
            bovelkedo++;
            if (i % 6 == 0) {
                bovelkedo2++;
            }
        }
    }
    printf("Tokeletes szamok 1 es %d kozott: %d\n", N1, tokeletes);
    printf("Hianyos szamok 1 es %d kozott: %d\n", N2, hianyos);
    printf("Bovelkedo szamok 1 es %d kozott: %d\n", N3, bovelkedo);
    printf("2-vel es 3-al oszthato bovelkedo szamok 1 es %d kozott:
%d\n", N3, bovelkedo2);
    return EXIT_SUCCESS;
}
```

3.1.1.

```
#include <stdio.h>
int main() {
    const int size=6;
    int idxI;
    char string[size+1]; // +1 for ending 0
    printf("string=");
    scanf("%6s", string);
    printf("The string is: %s\n", string);
    printf("The static string address, string: %p\n", string);
    printf("The dynamic string address, &string: %p\n", &string);
    for (idxI=0; idxI<size; idxI++) {
        printf("\'%c\'       ", string[idxI]);
    }
    printf("\n");
    for (idxI=0; idxI<size; idxI++) {
        if (string[idxI]==0) printf("0x");
        printf("%#-2x      ", string[idxI]);
    }
    printf("\n");
    for (idxI=0; idxI<size; idxI++) {
        printf("string[%d] ", idxI);
    }
    printf("\n");
    for (idxI=0; idxI<size; idxI++) {
        printf("%p  ", &(string[idxI]));
    }
    return 0;
}
```

3.1.2.

```
#include <stdio.h>
int main() {
    const int size = 6;
    int idxI;
    char string[size + 1]; // +1 for ending 0

    printf("string=");
    scanf("%6s", string);
    printf("The string is: %s\n", string);
    printf("The static string address, string: %p\n", string);
    printf("The dynamic string address, &string: %p\n", &string);
    for (idxI = 0; idxI < size; idxI++) {
        printf("\'%c\'", string[idxI]);
        if (string[idxI] == 0) {
            printf("\t0x");
            printf("%#-2x", string[idxI]);
        } else {
            printf("\t%#-2x", string[idxI]);
        }
        printf("\tstring[%d] ", idxI);
        printf("\t%p ", &(string[idxI]));
        printf("\n");
    }
    return 0;
}
```

3.2.1.

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

```
#include <math.h>
int main() {
    const int size=4;
    int idxI, op1[size]={2, -3, 0, 1},
    op2[size]={4, 2, -4, 5}, points=0, answer, result;
    for (idxI=0; idxI<size; idxI++) {
        printf("%d/%d=", op1[idxI], op2[idxI]);
        scanf("%d", &answer);
        result=op1[idxI]/op2[idxI];
        printf("%d/%d=%d\n", op1[idxI], op2[idxI], result);
        if (answer==result) {
            points++;
        }
        printf("%d%%%d=", op1[idxI], op2[idxI]);
        scanf("%d", &answer);
        result=op1[idxI]%op2[idxI];

        printf("%d%%%d=%d\n\n", op1[idxI], op2[idxI], result);
        if (answer==result) {
            points++;
        }
    }
    printf("You have %d points.\n", points);
    return 0;
}
```

3.2.2.

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

int main() {
    int A,B,TIP;
    double result;

    srand(time(NULL));
    A=((rand() % 20));
    B=((rand() % 20));
    TIP=(rand() % 2);
    if(!TIP){
        result=A/B;
        printf("Integer division:\n");
        printf("%d/%d=%d\n",A,B,(int)result);
    }else{
        result=(double)A/(double)B;
        printf("Floating point division:\n");
        printf("%d/%d=%.4lf\n",A,B,result);
    }
    return 0;
}
```

3.3.1.

```
#include <stdio.h>
int main() {
    char string[]="23 35678 1.2345e11 -9.8765432e-11";
    short int shortIntValue;
    int intValue;
    float floatValue;
    double doubleValue, sum;
```

```
    sscanf(string, "%hd %d %f %lf", &shortIntValue, &intValue,
&floatValue, &doubleValue);
    sum = shortIntValue + intValue + floatValue + doubleValue;
    sprintf(string, "%lf", sum);
    printf("The sum is %s", string);
    return 0;
}
```

3.3.2.

```
#include <stdio.h>
int main() {
    int idxI;
    char string[] = "48.1658745 -654686835454.321635432435421 -
9.8765432e-11";
    double doubleValue[3], sum = 0;

    sscanf(string, "%lf %lf %lf", &doubleValue[0], &doubleValue[1],
&doubleValue[2]);
    for (idxI = 0; idxI < 3; idxI++) {
        doubleValue[idxI] = doubleValue[idxI] +idxI +1;
        sum += doubleValue[idxI];
    }
    sprintf(string, "%+.8lf %+.8lf %+.8lf", doubleValue[0],
doubleValue[1], doubleValue[2]);
    printf("The new string array is %s\n", string);
    return 0;
}
```

3.4.1.

```
#include <stdio.h>
#include <conio.h>
int main() {
    char string1[100], string2[100], string3[100];
    int c, idxI;

    printf("String input with getchar. Press enter, ctrl+z and enter to
end input!\n");
    idxI=0;
    while((c=getchar()) != EOF) {
        string1[idxI]=c;
        idxI++;
    }
    string1[--idxI]=0;
    printf("String input with gets: ");
    gets(string2);
    printf("String input with scanf: ");
    scanf(„%s”, string3);
    printf("\nstring1 with getchar = %s\n", string1);
    printf("string2 with gets = %s\n", string2);
    printf("string3 with scanf = %s\n", string3);
    printf("Press any key to exit!");
    c=getch();
    return 0;
}
```

3.4.2.

A scanf a bufferben hagyja az entert így a gets nem áll meg, hanem az üres sort veszi át a veremből. Egy plusz getchar a scanf előtt kiveszi az entert a bufferből.

3.5.1

```
#include <stdio.h>
#include <math.h>
int main() {
    float tomb[5];
    int idxI;

    for (idxI=0; idxI<5; idxI++) {
        tomb[idxI]=sqrt(idxI);
    }
    for (idxI=0; idxI<5; idxI++) {
        printf("%f, ", tomb[idxI]);
    }
    return 0;
}
```

3.5.2.

```
#include <stdio.h>
#include <math.h>
int main() {
    float * tomb;
    int idxI, size;

    printf("Size of array? ");
    scanf("%d", &size);
    tomb = (float*) malloc(size*sizeof(float));
    for (idxI = 0; idxI < size; idxI++) //tomb feltoltese
        tomb[idxI] = sqrt(idxI);
    for (idxI = 0; idxI < size; idxI++) { //tomb kiiratasa
        printf("%f", tomb[idxI]);
        if(idxI+1 != size)
            printf(",");
        else
            printf("\n");
    }
    free(tomb);
    return 0;
}
```

3.6.1.

```
#include <stdio.h>
#include <math.h>
int main() {
    const int size=4;
    int tomb[size];
    int idxI, idxJ;

    for (idxI=0; idxI<size; idxI++) {
        printf("The %d. value: ", idxI);
        scanf("%d", &tomb[idxI]);
    }
    printf("\n%20.20s", " ");
    for (idxI=0; idxI<size; idxI++) {
```

```
        for (idxJ=0; idxJ<12; idxJ++) {
            printf("*");
        }
        printf("");
    }
    printf("\n%20.20s", " ");
    for (idxI=0; idxI<size; idxI++) {
        printf("*%10d* ", tomb[idxI]);
    }
    printf("\n%20.20s", " ");
    for (idxI=0; idxI<size; idxI++) {
        for (idxJ=0; idxJ<12; idxJ++) {
            printf("*");
        }
        printf("");
    }
    printf("\n%20.20s", "index: ");
    for (idxI=0; idxI<size; idxI++) {
        printf("%-13d", idxI);
    }
    printf("\n%20.20s", "address: ");
    for (idxI=0; idxI<size; idxI++) {
        printf("%-#13p", &tomb[idxI]);
    }
    printf("\n%20.20s%#p", "tomb: ", tomb);
    printf("\n%20.20s%#p", „&tomb: ", &tomb);
    printf("\n%20.20s%d", "tomb size: ", size);
    return 0;
}
```

3.6.2.

A dinamikus tömb mint mutató és az ő címe nem egyezik meg.

3.7.1.

```
#include <stdio.h>
#include <float.h>
int main() {
    const int xSize=10, ySize=10;
    float matrix[xSize][ySize]={
        {4, 5, 6, 4, 2, 3, 4, 2, 4, 2},
        {2, 2, 6, 5, 8, 7, 5, 3, 4, 2},
        {4, 3, 6, 2, 6, 3, 4, 6, 7, 2},
        {7, 2, 6, -2, -3, -3, -4, 6, 1, 2},
        {4, 1, 6, -7, -2, -3, -2, -8, 4, 2},
        {8, 1, 6, -7, -4, -3, -7, -6, -4, -2},
        {4, 3, 6, 7, 3, -3, -4, -8, -2, -2},
        {9, 2, 6, 6, 2, 2, -4, -8, -4, 2},
        {4, 3, 6, 8, 3, 2, 4, -6, 4, 2},
        {3, 3, 6, 7, 1, 3, 5, 6, 4, 2}};
    int water, land, biggestIndexX, biggestIndexY, biggestWaterIndexX,
    biggestWaterIndexY, idxI, idxJ;
    float mostShallowWater, sum, avg;

    water=0;
    land=0;
    for (idxI=0; idxI<xSize; idxI++) {
        for (idxJ=0; idxJ<ySize; idxJ++) {
            if (matrix[idxI][idxJ]>0) land++;
            else water++;
```

```
        }
    }
    printf("The ration to land to water is = %d:%d\n", land, water);

    biggestIndexX=0;
    biggestIndexY=0;
    for (idxI=0; idxI<xSize; idxI++) {
        for (idxJ=0; idxJ<ySize; idxJ++) {
            if (matrix[idxI][idxJ] > matrix[biggestIndexX][biggestIndexY])
{
                biggestIndexX=idxI;
                biggestIndexY=idxJ;
            }
        }
    }
    printf("The heighest point is at row: %d, column: %d and its height
is: %f\n", biggestIndexX, biggestIndexY,
matrix[biggestIndexX][biggestIndexY]);

    // conditional optimum search
    biggestWaterIndexX=-1;
    biggestWaterIndexY=-1;
    mostShallowWater=-FLT_MAX;
    for (idxI=0; idxI<xSize; idxI++) {
        for (idxJ=0; idxJ<ySize; idxJ++) {
            if (matrix[idxI][idxJ] < 0 && matrix[idxI][idxJ] >
mostShallowWater) {
                biggestWaterIndexX=idxI;
                biggestWaterIndexY=idxJ;

    mostShallowWater=matrix[biggestWaterIndexX][biggestWaterIndexY];
            }
        }
    }
    if (biggestWaterIndexX == -1) {
        printf("There is no water so the search is meaningless.\n");
    } else {
        printf("The most shallow water is at row: %d, column: %d and its
depth is: %f\n", biggestWaterIndexX, biggestWaterIndexY,
mostShallowWater);
    }

    sum=0;
    for (idxI=0; idxI<xSize; idxI++) {
        for (idxJ=0; idxJ<ySize; idxJ++) {
            sum+=matrix[idxI][idxJ];
        }
    }
    avg=sum /(xSize*ySize);
    printf("The average height is: %f\n", avg);

    return 0;
}
```

3.7.2.

```
#include <stdio.h>
#include <float.h>
int main() {
    const int xSize = 10, ySize = 10;
```

```
    float matrix[10][10] = {
        {4, 5, 6, 4, 2, 3, 4, 2, 4, 2},
        {2, 2, 6, 5, 8, 7, 5, 3, 4, 2},
        {4, 3, 6, 2, 6, 3, 4, 6, 7, 2},
        {7, 2, 6, -2, -3, -3, -4, 6, 1, 2},
        {4, 1, 6, -7, -2, -3, -2, -8, 4, 2},
        {8, 1, 6, -7, -4, -3, -7, -6, -4, -2},
        {4, 3, 6, 7, 3, -3, -4, -8, -2, -2},
        {9, 2, 6, 6, 2, 2, -4, -8, -4, 2},
        {4, 3, 6, 8, 3, 2, 4, -6, 4, 2},
        {3, 3, 6, 7, 1, 3, 5, 6, 4, 2}
    };
    int biggestWaterIndexX, biggestWaterIndexY, lowestWaterIndexX,
lowestWaterIndexY;
    int biggestLandIndexX, biggestLandIndexY, lowestLandIndexX,
lowestLandIndexY;
    int idxI, idxJ;
    float highestLand=-1, lowestLand=-1, deepestWater = 1,
shallowestWater = 1;

    for (idxI=0; idxI<xSize && highestLand == -1; idxI++) {
//meghatarozzuk, hogy van-e fold
        for (idxJ=0; idxJ<ySize && highestLand == -1; idxJ++) {
            if (matrix[idxI][idxJ]>0)
                highestLand = lowestLand = matrix[idxI][idxJ];
        }
    }
    for (idxI=0; idxI<xSize && deepestWater == 1; idxI++) {
//meghatarozzuk, hogy van-e tenger
        for (idxJ=0; idxJ<ySize && deepestWater == 1; idxJ++) {
            if (matrix[idxI][idxJ]<=0)
                deepestWater = shallowestWater = matrix[idxI][idxJ];
        }
    }
    //foldon es vizen a legnagyobb es legkisebb ertekek meghatarozasa
    for (idxI = 0; idxI < xSize; idxI++) {
        for (idxJ = 0; idxJ < ySize; idxJ++) {
            if (matrix[idxI][idxJ] > 0) {
                if (matrix[idxI][idxJ] > highestLand)
                    highestLand = matrix[idxI][idxJ];
                else if (matrix[idxI][idxJ] < lowestLand)
                    lowestLand = matrix[idxI][idxJ];
            }
            else {
                if (matrix[idxI][idxJ] < deepestWater)
                    deepestWater = matrix[idxI][idxJ];
                else if (matrix[idxI][idxJ] > shallowestWater)
                    shallowestWater = matrix[idxI][idxJ];
            }
        }
    }
    if(highestLand != -1)
        printf("The level difference on land is %f\n", highestLand -
lowestLand);
    if (deepestWater != 1)
        printf("The level difference in water is %f\n", shallowestWater -
deepestWater);
    if (highestLand != -1 && deepestWater != 1)
```

```
        printf("The biggest global level difference is %f\n", highestLand
- deepestWater);
    else
        printf("There's no global level difference!\n");
    return 0;
}
```

3.8.1.

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

int main() {
    int repeatRequired, repeatAct, num, trial, idxI;
    int frequency[6]={0};

    printf("How much consecutive 6 do you want: ");
    scanf("%d", &repeatRequired);
    srand((unsigned)time(NULL));
    repeatAct=0;
    trial=0;
    while (repeatAct!=repeatRequired) {
        num = (rand()%6)+1;
        frequency[num-1]++;
        if (num==6) repeatAct++;
        else repeatAct=0;
        trial++;
    }
    printf("It needed %d random number to generate %d consecutive
6.\n\n", trial, repeatRequired);
    printf("Meanwhile we generated\n");
    for (idxI=0; idxI<6; idxI++)
        printf("%d pieces of %d\n", frequency[idxI], idxI+1);
    // repeatRequired=11, 100 millio trial, 2 s  (2006)
    return 0;
}
```

3.8.2.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main() {
    int repeatRequired, repeatAct, num, trial, idxI;
    int frequency[6] = {0}, sequence[30];
    long int sum = 0;

    printf("How much consecutive 6 do you want: ");
    scanf("%d", &repeatRequired);
    srand((unsigned)time(NULL));

    for (idxI = 0; idxI < 30; ++idxI) {
        repeatAct = 0;
        trial = 0;
        while (repeatAct!=repeatRequired) {
            num = (rand()%6) + 1;
            if (num == 6) //ha hatos akkor szamolom hany db egymas utan 6-
os
                repeatAct++;
            else //ha valamelyik szam nem hatos, ujrakezdem a szamolast
```

```
                repeatAct=0;
            trial++;
        }
        sequence[idxI] = trial; //hany probalkozas kellett az 1. 2. 3...
ismetlesnel
    }
    for (idxI = 0; idxI < 30; ++idxI) {
        sum += sequence[idxI];
    }
    printf("It's needed on average %d random number to generate %d
consecutive 6.\n\n", sum/30, repeatRequired);
    return 0;
}
```

3.9.1.

```
#include <stdio.h>

void print(float a, float b, float c) {
    printf("a=%f b=%f c=%f\n", a, b, c);
}
void shift(float* a, float* b, float* c, int cyclic) {
    float temp;

    temp = *c;
    *c = *b;
    *b = *a;
    if (cyclic)
        *a = temp;
    else
        *a = 0;
}

int main() {
    float x1, x2, x3;

    printf("Provide the next number: ");
    scanf("%f", &x1);
    printf("Provide the next number: ");
    scanf("%f", &x2);
    printf("Provide the next number: ");
    scanf("%f", &x3);
    printf("Original: ");
    print(x1, x2, x3);
    shift(&x1, &x2, &x3, 1);
    printf("After cyclic shift: ");
    print(x1, x2, x3);
    shift(&x1, &x2, &x3, 0);
    printf("After non-cyclic shift: ");
    print(x1, x2, x3);
    return 0;
}
```

3.9.2.

```
#include <stdio.h>

void print(float a[], int size) {
    int i;
    for(i = 0; i < size; ++i)
        printf("a[%d]= %f ", i, a[i]);
```

```
    printf("\n");
}
void shift(float* a, int cyclic) {
    float temp;

    temp = a[2];
    a[2] = a[1];
    a[1] = a[0];
    if (cyclic)
        a[0] = temp;
    else
        a[0] = 0;
}

int main() {
    float x[3];

    printf("Provide the next number: ");
    scanf("%f", &x[0]);
    printf("Provide the next number: ");
    scanf("%f", &x[1]);
    printf("Provide the next number: ");
    scanf("%f", &x[2]);
    printf("Original: ");
    print(x, 3);
    shift(x, 1);
    printf("After cyclic shift: ");
    print(x, 3);
    shift(x, 0);
    printf("After non-cyclic shift: ");
    print(x, 3);
    return 0;
}
```

3.9.3.

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

void print(float a[], int size) {
    int idxI;
    for(idxI = 0; idxI < size; ++idxI)
        printf("myArray[%d]= %f ", idxI, a[idxI]);
    printf("\n");
}
void shift(float* a, char direction, int cyclic, int size) {
    float temp = (direction=='r')?a[size-1]:a[0];
    int idxI;
    if(direction == 'r') { //ha jobbra shiftelunk
        for(idxI = size-1; idxI > 0; --idxI) {
            a[idxI] = a[idxI-1];
        }
        if (cyclic)
            a[0] = temp;
        else
            a[0] = 0;
    }
    else { //ha balra shiftelunk
        for(idxI = 0; idxI < size; ++idxI) {
            a[idxI] = a[idxI+1];
```

```
        }
        if (cyclic)
            a[size-1] = temp;
        else
            a[size-1] = 0;
    }
}

int main() {
    float *myArray;
    int size, idxI;
    char direction;

    printf("Array size? "); scanf("%d", &size);
    myArray = (float*) malloc(sizeof(float)*size);
    for(idxI = 0; idxI < size; ++idxI) {
        printf("Provide the next number: ");
        scanf("%f", &myArray[idxI]);
    }
    _flushall();
    printf(„Original array: „);
    print(myArray, size);
    printf("In which direction do you want to shift? (l/r) ");
scanf("%c", &direction);
    _flushall();

    printf("After cyclic shift: ");
    shift(myArray, direction, 1, size);
    print(myArray, size);

    printf("After non-cyclic shift: ");
    shift(myArray, direction, 0, size);
    print(myArray, size);
    return 0;
}
```

3.10.1

```
/* FILE INC_HPP_*/
#define INC_HPP_

void print(double*, int);
double sum(double*, int);
double avg(double*, int);
int minIndex(double*, int);
int searchFor(double*, int, double);

#endif


/* FILE INC_CPP_*/
#include <stdio.h>
#include "inc.hpp"

void print(double* tomb, int size) {
    int idxI;
    printf("[");
    for (idxI=0; idxI<size; idxI++)
        printf("%lf ", tomb[idxI]);
    printf("]");
```

```
}
double sum(double* tomb, int size) {
    double result=0;
    int idxI;
    for (idxI=0; idxI<size; idxI++)
        result += tomb[idxI];
    return result;
}
double avg(double* tomb, int size) {
    double result = sum(tomb, size) / size;
    return result;
}
int minIndex(double* tomb, int size) {
    int minIndex = 0;
    int idxI;
    for (idxI=1; idxI<size; idxI++)
        if (tomb[idxI] < tomb[minIndex])
            minIndex = idxI;
    return minIndex;
}
int searchFor(double* tomb, int size, double data) {
    int idxI;
    for (idxI=1; idxI<size; idxI++)
        if (tomb[idxI] == data)
            return 1; // data found
    return 0; // data is not found
}


/* FILE ARRAY_CPP_*/
#include <stdio.h>
#include "inc.hpp"

int main() {
    double myArray[]={12, 56, -7}, mySum, myAvg;
    int size = sizeof myArray / sizeof (double); // works only with
static arrays
    double what=-8;

    print(myArray, size);
    mySum=sum(myArray, size);
    myAvg=avg(myArray, size);
    printf("\nThe sum of the array is: %lf, and the average is: %lf",
mySum, myAvg);
    printf("\nThe minimum elem is: %lf", myArray[minIndex(myArray,
size)]);
    if ( searchFor(myArray, size, what))
        printf("\n%lf is found.", what);
    else
        printf("\n%lf is not found.", what);
    return 0;
}
```

3.10.2.

```
#ifndef FUNCTION_H_
#define FUNCTION_H_

void print(double*, int);
```

```
double sum(double*, int);
double avg(double*, int);
int minIndex(double*, int);
int maxIndex(double*, int);
void change(double*, double*, int);
int searchFor(double*, int, double);

#endif /* FUNCTION_H_ */

/* FILE FUNCTION_C*/
#include <stdio.h>
#include "function.h"

void print(double* tomb, int size) {
    int idxI;
    printf("[");
    for (idxI = 0; idxI < size; idxI++)
        printf("%lf ", tomb[idxI]);
    printf("]");

}
double sum(double* tomb, int size) {
    double result = 0;
    int idxI;
    for (idxI = 0; idxI < size; idxI++)
        result += tomb[idxI];
    return result;
}
double avg(double* tomb, int size) {
    double result = sum(tomb, size) / size;
    return result;
}
int minIndex(double* tomb, int size) {
    int minIndex = 0;
    int idxI;
    for (idxI = 1; idxI < size; idxI++)
        if (tomb[idxI] < tomb[minIndex])
            minIndex = idxI;
    return minIndex;
}

int maxIndex(double* tomb, int size) {
    int maxIndex = 0;
    int idxI;
    for (idxI = 1; idxI < size; idxI++)
        if (tomb[idxI] > tomb[maxIndex])
            maxIndex = idxI;
    return maxIndex;
}

void change(double* tomb, double* tomb2, int size) {
    double helper;
    int idxI;
    for (idxI = 0; idxI < size; idxI++) {
        helper = tomb[idxI];
        tomb[idxI] = tomb2[idxI];
        tomb2[idxI] = helper;
    }
```

```
}

int searchFor(double* tomb, int size, double data) {
    int idxI;
    for (idxI = 1; idxI < size; idxI++)
        if (tomb[idxI] == data)
            return 1; // data found
    return 0; // data is not found
}


/* FILE 2_35_2_C*/
#include <stdio.h>
#include "function.h"

int main() {
    double myArray[] = { 12, 56, -7 }, myArray2[] = { 24, -26, 4 },
mySum, myAvg;
    int size = sizeof myArray / sizeof(double); // works only with static
arrays
    double what = -8;

    print(myArray, size);
    mySum = sum(myArray, size);
    myAvg = avg(myArray, size);
    printf("\nThe sum of the array is: %lf, and the average is: %lf",
mySum,
           myAvg);
    printf("\nThe minimum elem is: %lf", myArray[minIndex(myArray,
size)]);
    printf("\nThe maximum elem index is: %d", maxIndex(myArray, size));
    printf("\nThe old array:\nA: ");
    print(myArray, size);
    printf("\nB: ");
    print(myArray2, size);
    change(myArray, myArray2, size);
    printf("\nThe new array:\nA: ");
    print(myArray, size);
    printf("\nB:");
    print(myArray2, size);
    if (searchFor(myArray, size, what))
        printf("\n%lf is found.", what);
    else
        printf("\n%lf is not found. ", what);
    return 0;
}
```

3.10.3.

```
#ifndef FUNCTION_H_
#define FUNCTION_H_

void print(double*, int);
double sum(double*, int);
double avg(double*, int);
int minIndex(double*, int);
int maxIndex(double*, int);
void change(double*, double*, int);
void alapmuvelet(double*, double*, int);
double skalaris(double*, double*, int);
```

```
int elsoElofordulas(double*, int, double);
int utolsoElofordulas(double*, int, double);
int searchFor(double*, int, double);

#endif /* FUNCTION_H_ */

/* FILE FUNCTION_C*/
#include <stdio.h>
#include "function.h"

void print(double* tomb, int size) {
    int idxI;
    printf("[");
    for (idxI = 0; idxI < size; idxI++)
        printf("%lf ", tomb[idxI]);
    printf("]");

}
double sum(double* tomb, int size) {
    double result = 0;
    int idxI;
    for (idxI = 0; idxI < size; idxI++)
        result += tomb[idxI];
    return result;
}
double avg(double* tomb, int size) {
    double result = sum(tomb, size) / size;
    return result;
}
int minIndex(double* tomb, int size) {
    int minIndex = 0;
    int idxI;
    for (idxI = 1; idxI < size; idxI++)
        if (tomb[idxI] < tomb[minIndex])
            minIndex = idxI;
    return minIndex;
}
int maxIndex(double* tomb, int size) {
    int maxIndex = 0;
    int idxI;
    for (idxI = 1; idxI < size; idxI++)
        if (tomb[idxI] > tomb[maxIndex])
            maxIndex = idxI;
    return maxIndex;
}
void change(double* tomb, double* tomb2, int size) {
    double helper;
    int idxI;
    for (idxI = 0; idxI < size; idxI++) {
        helper = tomb[idxI];
        tomb[idxI] = tomb2[idxI];
        tomb2[idxI] = helper;
    }

}
void alapmuvelet(double* tomb, double*tomb2, int size) {
    int idxI;
    for (idxI = 0; idxI < size; idxI++) {
        printf("\n%d.", idxI);
```

```
        printf("\n%lf + %lf = %lf", tomb[idxI], tomb2[idxI], tomb[idxI]
                + tomb2[idxI]);
        printf("\n%lf - %lf = %lf", tomb[idxI], tomb2[idxI], tomb[idxI]
                - tomb2[idxI]);
        printf("\n%lf * %lf = %lf", tomb[idxI], tomb2[idxI], tomb[idxI]
                * tomb2[idxI]);
        printf("\n%lf / %lf = %lf", tomb[idxI], tomb2[idxI], tomb[idxI]
                / tomb2[idxI]);
    }
}
double skalaris(double* tomb, double*tomb2, int size) {
    double skalar = 0;
    int idxI;
    for (idxI = 0; idxI < size; idxI++) {
        skalar += tomb[idxI] * tomb2[idxI];
    }
    return skalar;
}
int elsoElofordulas(double* tomb, int size, double data) {
    int index = -1;
    int idxI;
    for (idxI = 0; idxI < size; idxI++) {
        if ((index == -1) && (data == tomb[idxI])) {
            index = idxI;
        }
    }
    return index;
}
int utolsoElofordulas(double* tomb, int size, double data) {
    int index = 0;
    int idxI;
    for (idxI = 0; idxI < size; idxI++) {
        if (data == tomb[idxI]) {
            index = idxI;
        }
    }
    return index;
}
int searchFor(double* tomb, int size, double data) {
    int idxI;
    for (idxI = 1; idxI < size; idxI++)
        if (tomb[idxI] == data)
            return 1; // data found
    return 0; // data is not found
}

/* FILE 2_35_3_C*/
#include <stdio.h>
#include "function.h"

int main() {
    double myArray[] = { 12, 56, -7, 56 }, myArray2[] = { 24, -26, 4, 19
},
            mySum, myAvg;
    int size = sizeof myArray / sizeof(double), elsoindex, utolsoindex;
// works only with static arrays
    double what = -8, what2 = 56;

    print(myArray, size);
```

```
    mySum = sum(myArray, size);
    myAvg = avg(myArray, size);
    printf("\nThe sum of the array is: %lf, and the average is: %lf",
mySum,
           myAvg);
    printf("\nThe minimum elem is: %lf", myArray[minIndex(myArray,
size)]);
    printf("\nThe maximum elem index is: %d", maxIndex(myArray, size));
    if (searchFor(myArray, size, what))
        printf("\n%lf is found.", what);
    else
        printf("\n%lf is not found.", what);
    printf("\nThe old array:\nA:");
    print(myArray, size);
    printf("\nB:");
    print(myArray2, size);
    change(myArray, myArray2, size);
    printf("\nThe new array:\nA:");
    print(myArray, size);
    printf("\nB:");
    print(myArray2, size);
    alapmuvelet(myArray, myArray2, size);
    printf("\nSkaláris szorzat: %lf", skalaris(myArray, myArray2, size));
    elsoindex = elsoElofordulas(myArray2, size, what2);
    utolsoindex = utolsoElofordulas(myArray2, size, what2);
    if (elsoindex != -1) {
        printf("\n%lf elso elofordulas indexe: %d", what2, elsoindex);
        printf("\n%lf utolso elofordulas indexe: %d", what2, utolsoindex);
    } else {
        printf("\n%lf nem talalhato! ", what2);
    }
    return 0;
}
```

3.11.1.

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

int devisableBy4(int num) {
    if (num/4==num/4.)
        return 1;
    return 0;
}

int main() {
    char s[]="The project will scale up to 1,200 marine sites,\n"
     "including different conditions such as surface waters,\n"
     "waters near methane emissions from the sea floor, and deep-sea
sediments. ";

    int idxI, size;
    size = strlen(s);
    for (idxI=0; idxI<size; idxI++) {
        if (!devisableBy4(idxI) || (s[idxI]=='\n' || s[idxI]=='\t' ||
s[idxI]==' '))
            printf("%c", s[idxI]);
    }
    return 0;
}
```

3.11.2.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main() {
    char string[] = "The project will scale up to 1,200 marine sites,\n"
        "including different conditions such as surface waters,\n"
        "waters near methane emissions from the sea floor, and deep-
seasediments.";
    int idxI, size;
    size = strlen(string);

    for (idxI = 0; idxI < size; idxI++) {
        if ( (toupper(string[idxI]) == 'A' || toupper(string[idxI]) == 'E'
|| toupper(string[idxI]) == 'I' || toupper(string[idxI]) == 'O' ||
toupper(string[idxI])== 'U') && string[idxI - 1] != ' ')
                ; //ha maganhazo,  es elotte nem szokoz van, tehat nem
keudobetu, akkor nem csinal semmit
        else
            printf("%c", string[idxI]);
    }
    return 0;
}
```

3.12.1.

```
#include <stdio.h>
#include <string.h>
int main() {
    char sz[80];
    char c;
    int  n, a, o, i, e, u, idxI;
    printf("text=\n");
    gets(sz);
    n = strlen(sz);
    a = o = i = e = u = 0;
    for(idxI = 0; idxI < n; idxI++) {
        c = sz[idxI];
        switch (c) {
            case 'a':
            case 'A':
                a++;
                break;
            case 'o':
            case 'O':
                o++;
                break;
            case 'i':
            case 'I':
                i++;
                break;
            case 'e':
            case 'E':
                e++;
                break;
            case 'u':
            case 'U':
                u++;
                break;
        }
```

```
    }
    printf("Vowel statistics:\n");
    printf("   a,A: %d\n", a);
    printf("   e,E: %d\n", e);
    printf("   i,I: %d\n", i);
    printf("   o,O: %d\n", o);
    printf("   u,U: %d\n", u);
    return 0;
}
```

3.12.2.

```
#include <stdio.h>
#include <string.h>
int main() {
    char text[80];
    char nextChar;
    int textSize, sentence, word, idxI;

    printf("Text= ");
    gets(text);
    textSize = strlen(text);
    if (text[0] == '\0')
        sentence = word = 0;
    else {
        sentence = 0;
        word = 1;
    }
    for (idxI = 0; idxI < textSize; idxI++) {
        nextChar = text[idxI];
        switch (nextChar) {
        case ' ':
            word++;
            break;
        case '.':
        case '!':
        case '?':
            sentence++;
            break;
        }
    }
    printf("Vowel statistics:\n");
    printf("Word: %d\n", word);
    printf("Sentence: %d\n", sentence);
    return 0;
}
```

3.13.1.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int in(char c) {
    char s[]="AEIOU";
    unsigned int idxI;
    idxI=0;
    do {
        if (c == s[idxI]) return 1;
        idxI++;
    } while (idxI <= strlen(s));
```

```
    return 0;
}
int main() {
    char orig[255], newVer[255], kar;
    unsigned int idxI;
    int idxJ;
    printf("sentence=\n");
    gets(orig);
    idxJ=-1;
    for(idxI=0; idxI<strlen(orig); idxI++) {
        idxJ++;
        kar=toupper(orig[idxI]);
        newVer[idxJ]=kar;
        if( in(kar) ) {
            newVer[++idxJ]='V';
            newVer[++idxJ]=kar;
        }
    }
    newVer[++idxJ]='\0';
    printf("%s", newVer);
    return 0;
}
```

3.13.2.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <malloc.h>
int main() {
    char original[255], *newVer;
    int idxI, idxJ;

    printf("Sentence: ");
    gets(original);
    newVer = (char*) malloc (sizeof(char)*strlen(original)); //dinamikus
foglalas
    idxJ = 0;
    for (idxI = 0; idxI < strlen(original); idxI++) {
        if (toupper(original[idxI]) != 'V') {
            newVer[idxJ] = toupper(original[idxI]);
            idxJ++;
        }
        else { //ha v betut talalok
            if ( toupper(original[idxI]) == toupper(original[idxI + 2]) )
{ //ha egymas utan ket v; vivi->vi
                newVer[idxJ++] = toupper(original[idxI++]);
                newVer[idxJ++] = toupper(original[idxI++]);
                idxI++;
            }
            else
                idxI++;
        }
    }
    newVer[idxJ] = '\0';
    printf("%s\n", newVer);
    free(newVer);
    return 0;
}
```

3.14.1.

```
#include <stdio.h>
#include <string.h>
void substract(char* from, char* what) {
    char* subString=0;
    int len;

    subString=strstr(from, what);
    if (!subString) return;
    len = strlen(what);
    strcpy(subString, subString+len); // almafavirag -> almavirag\0g
}
int main() {
    char from[200], what[100];

    printf("Substract from=");
    scanf("%s", from);
    printf("Substract what=");
    scanf("%s", what);
    substract(from, what);
    printf("\nThe reduced word is: %s", from);
    return 0;
}
```

3.14.2.

```
#include <stdio.h>
#include <string.h>
void substract(char* from, char* what) {
    char* subString = 0;
    int length;
    subString = strstr(from, what);
    if (!subString)
        return;
    length = strlen(what);
    strcpy(subString, subString + length); // almafavirag -> almavirag\0g
    substract(from, what);
}
int main() {
    char from[200], what[100];

    printf("Substract from=");
    scanf("%s", from);
    printf("Substract what=");
    scanf("%s", what);
    substract(from, what);
    printf("\nThe reduced word is: %s\n", from);
    return 0;

}
```

3.15.1.

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

char* upperCase(char* string) {
    char* result=0;
    int len, idxI;
```

```
    result = strdup(string);
    len = strlen(string);
    for (idxI=0; idxI<len; idxI++)
        result[idxI]=toupper(result[idxI]);
    return result;
}
char* lowerCase(char* string) {
    char* result=0;
    int len, idxI;

    result = strdup(string);
    len = strlen(string);
    for (idxI=0; idxI<len; idxI++)
        result[idxI]=tolower(result[idxI]);
    return result;
}
char* toggleCase(char* string) {
    char* result=0;
    int len, idxI;

    result = strdup(string);
    len = strlen(string);
    for (idxI=0; idxI<len; idxI++) {
        if (isupper(result[idxI]))
            result[idxI]=tolower(result[idxI]);
        else
            result[idxI]=toupper(result[idxI]);
    }
    return result;
}
char* sentenceCase(char* string) {
    char* result=0;
    int len, idxI;

    result = strdup(string);
    len = strlen(string);
    result[0]=toupper(result[0]);
    for (idxI=1; idxI<len; idxI++)
        result[idxI]=tolower(result[idxI]);
    return result;
}
char* titleCase(char* string) {
    char* result=0;
    int len, idxI, startWord=1;

    result = strdup(string);
    len = strlen(string);
    for (idxI=0; idxI<len; idxI++) {
        if (startWord)
            result[idxI]=toupper(result[idxI]);
        else
            result[idxI]=tolower(result[idxI]);
        if (result[idxI]==' ')
            startWord = 1;
        else
            startWord = 0;
    }
    return result;
```

```
}
int menu() {
    int select;
    printf("UPPER CASE - 1\n");
    printf("lower case - 2\n");
    printf("tOGGLE CASE - 3\n");
    printf("Sentence case - 4\n");
    printf("Title Case - 5\n");
    scanf("%d", &select);
    return select;
}
int main() {
    char myString[200], *result;
    int cheoice;
    printf("orinigal string=");
    gets(myString);
    choice = menu();
    switch (choice) {
        case 1:
            result = upperCase(myString);
            break;
        case 2:
            result = lowerCase(myString);
            break;
        case 3:
            result = toggleCase(myString);
            break;
        case 4:
            result = sentenceCase(myString);
            break;
        case 5:
            result = titleCase(myString);
            break;
    }
    printf("\nresult=\"%s\"", result);
    free(result);
    return 0;
}
```

3.15.2.

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

char* lowerCase(char* string) {
    char* result=0;
    int len, idxI;

    result = _strdup(string);
    len = strlen(string);
    for (idxI = 0; idxI < len; idxI++)
        result[idxI] = tolower(result[idxI]);
    return result;
}
char* sentenceCase(char* string) {
    char* result = 0;
    int len, idxI;
```

```
    result = _strdup(string);
    len = strlen(string);
    result[0] = toupper(result[0]);
    for (idxI = 1; idxI < len; idxI++)
        result[idxI] = tolower(result[idxI]);
    return result;
}
char* niceTitleCase(char* string) {
    char* result = 0;
    int len, idxI, startWord = 1;
    result = _strdup(string);
    len = strlen(string);

    result = lowerCase(result); //elosszor minden betu legyen kicsi
    result = sentenceCase(result); //a mondat kezdodjon nagybetuvel
    for (idxI = 0; idxI < len; idxI++) {
        if (result[idxI] == ':') { //ha :-ot talalunk a kovetkezo betu
legyen mindenfele keppen nagybetu
            result[idxI+1] = toupper(result[idxI+1]);
            idxI++;
        }
         //for, and, the, from, of, in... szavak maradjanak kisbetusek
        else if( result[idxI] == 'f' && result[idxI+1] == 'o' &&
result[idxI+2] == 'r' && result[idxI+3] == ' ')
            idxI += 3;
        else if (result[idxI] == 'a' && result[idxI+1] == 'n' &&
result[idxI+2] == 'd' && result[idxI+3] == ' ')
            idxI += 3;
        else if (result[idxI] == 't' && result[idxI+1] == 'h' &&
result[idxI+2] == 'e' && result[idxI+3] == ' ')
            idxI += 3;
        else if (result[idxI] == 'f' && result[idxI+1] == 'r' &&
result[idxI+2] == 'o' && result[idxI+3] == 'm' && result[idxI+4] == ' ')
            idxI += 4;
        else if (result[idxI] == 'o' && result[idxI+1] == 'f' &&
result[idxI+2] == ' ')
            idxI += 2;
        else if (result[idxI] == 'i' && result[idxI+1] == 'n' &&
result[idxI+2] == ' ')
            idxI += 2;
         //egyebkent minden szo kezdodjon nagybetuvel
        else if (startWord)
            result[idxI] = toupper(result[idxI]);
        else
            result[idxI] = tolower(result[idxI]);
        if (result[idxI] == ' ' || result[idxI] == '\t' || result[idxI] ==
'\n')
            startWord = 1;
        else
            startWord = 0;
    }
    return result;
}
int main() {
    char myString[200], *result;
    printf("original string=");
    gets(myString);

    result = niceTitleCase(myString);
```

```
    printf("\nresult= \"%s\"\n", result);
    free(result);
    return 0;
}
```

3.16.1.

```
#include <malloc.h>
#include <memory.h>

int main() {
    int size, idxI;
    double *array=NULL, *temp=NULL;

    printf("array size: ");
    scanf("%d", &size);
    array = (double*)malloc(size*sizeof(double));
    for (idxI=0; idxI<size; idxI++) {
        printf("%d. element: ", idxI);
        scanf("%lf", &array[idxI]);
    }
    temp = (double*)malloc(size*2*sizeof(double));
    memcpy(temp, array, size*sizeof(double));
    array = temp;
    size *= 2;
    temp = NULL; // free is not needed here
    for (idxI=size/2; idxI<size; idxI++) {
        printf("%d. element: ", idxI);
        scanf("%lf", &array[idxI]);
    }
    free(array);
    array = NULL;
    return 0;
}
```

3.16.2.

```
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
double* allocArray(int* size) {
    // version: 1 of 4
    double* result=NULL;
    printf("\narray size: ");
    scanf("%d", size);
    result = (double*)malloc(*size * sizeof(double));
    return result;
}
int main() {
    int size, idxI;
    double *array=NULL, *temp=NULL;

    array = allocArray(&size);
    for (idxI=0; idxI<size; idxI++) {
        printf("%d. element: ", idxI);
        scanf("%lf", &array[idxI]);
    }
    temp = (double*)malloc(size*2*sizeof(double));
    memcpy(temp, array, size*sizeof(double));
    array = temp;
```

```
    size *= 2;
    temp = NULL; // free is not needed here
    for (idxI=size/2; idxI<size; idxI++) {
        printf("%d. element: ", idxI);
        scanf("%lf", &array[idxI]);
    }
    free(array);
    array = NULL;
    return 0;
}
```

3.17.1.

```
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <time.h>

double** allocMatrix(int rows, int cols) {
    double** result=NULL;
    int idxI;
    if (rows<=0 || cols<=0)
        return NULL;
    result = (double**)malloc(rows * sizeof(double*));
    for (idxI=0; idxI<rows; idxI++) {
        result[idxI] = (double*)malloc(cols * sizeof(double));
    }
    return result;
}

void freeMatrix(double** mat, int rows) {
    int idxI;
    if (rows<=0)
        return;
    if (!mat)
        return;
    for (idxI=0; idxI<rows; idxI++) {
        if (!mat[idxI])
            free(mat[idxI]);
    }
    free(mat);
}

void randMatrix(double** mat, int rows, int cols, int min, int max) {
    int idxI, idxJ;
    if (rows<=0 || cols<=0)
        return;
    srand((unsigned int)time(NULL));
    for (idxI=0; idxI<rows; idxI++) {
        for (idxJ=0; idxJ<cols; idxJ++) {
            mat[idxI][idxJ] = rand()%(max-min+1)+min;
        }
    }
}

double** multMatrix(double** matA, int rowA, int colA, double** matB, int
rowB, int colB, int *rowC, int *colC) {
    double** result=NULL, sum;
    int idxI, idxJ, idxK;
    if (colA != rowB)
```

```
        return NULL;
    *rowC = rowA;
    *colC = colB;
    result = allocMatrix(*rowC, *colC);
    srand((unsigned int)time(NULL));
    for (idxI=0; idxI<rowA; idxI++) {
        for (idxJ=0; idxJ<colB; idxJ++) {
            sum = 0;
            for (idxK=0; idxK<colA; idxK++) {
                sum += matA[idxI][idxK]*matB[idxK][idxJ];
            }
            result[idxI][idxJ] = sum;
        }
    }
    return result;
}

void printMatrix(double** mat, int rows, int cols) {
    int idxI, idxJ;
    if (rows<=0 || cols<=0)
        return;
    for (idxI=0; idxI<rows; idxI++) {
        printf("[");
        for (idxJ=0; idxJ<cols; idxJ++) {
            printf("%lf, ", mat[idxI][idxJ]);
        }
        printf("]\n");
    }
}

int main() {
    int rowA, colA, rowB, colB, rowC, colC;
    double **matA=NULL, **matB=NULL, **matC=NULL;

    printf("Number of rows of matrix A: ");
    scanf("%d", &rowA);
    printf("Number of columns of matrix A: ");
    scanf("%d", &colA);
    printf("Number of rows of matrix B: ");
    scanf("%d", &rowB);
    printf("Number of columns of matrix B: ");
    scanf("%d", &colB);

    matA = allocMatrix(rowA, colA);
    matB = allocMatrix(rowB, colB);
    randMatrix(matA, rowA, colA, 12, 16);
    randMatrix(matB, rowB, colB, -4, 5);
    matC = multMatrix(matA, rowA, colA, matB, rowB, colB, &rowC, &colC);

    printf("matrix A =\n");
    printMatrix(matA, rowA, colA);
    printf("matrix B =\n");
    printMatrix(matB, rowB, colB);
    printf("matrix A*B =\n");
    printMatrix(matC, rowC, colC);

    freeMatrix(matA, rowA); matA = NULL;
    freeMatrix(matB, rowB); matB = NULL;
    freeMatrix(matC, rowC); matC = NULL;
```

```
    return 0;
}
```

3.17.2.

```
#include <stdio.h>
#include <malloc.h>
#include <memory.h>
#include <time.h>

double** allocMatrix(int rows, int cols) {
    double** result = NULL;
    int idxI;
    if (rows <= 0 || cols <= 0)
        return NULL;
    result = (double**) malloc(rows * sizeof(double*));
    for (idxI = 0; idxI < rows; idxI++) {
        result[idxI] = (double*) malloc(cols * sizeof(double));
    }
    return result;
}

void freeMatrix(double** mat, int rows) {
    int idxI;
    if (rows <= 0)
        return;
    if (!mat)
        return;
    for (idxI = 0; idxI < rows; idxI++) {
        free(mat[idxI]);
    }
    free(mat);
}

void randMatrix(double** mat, int rows, int cols, int min, int max) {
    int idxI, idxJ;
    if (rows <= 0 || cols <= 0)
        return;
    srand((unsigned int) time(NULL));
    for (idxI = 0; idxI < rows; idxI++) {
        for (idxJ = 0; idxJ < cols; idxJ++) {
            mat[idxI][idxJ] = rand() % (max - min + 1) + min;
        }
    }
}

void printMatrix(double** mat, int rows, int cols) {
    int idxI, idxJ;

    if (rows <= 0 || cols <= 0)
        return;
    for (idxI = 0; idxI < rows; idxI++) {
        printf("[");
        for (idxJ = 0; idxJ < cols; idxJ++) {
            printf("%.2lf, ", mat[idxI][idxJ]);
        }
        printf("]\n");
    }

}
```

```
void cutMatrix(double ** mat, int rows, int cols) {
    double** result = NULL;
    int selected, newcols = 0, newrows = 0, i, j, db = 0;
    printf("1. Cut the bottom and add to the right side.\n");
    printf("2. Cut the bottom\n");
    printf("3. Add to the right side\n");
    printf("Choose: ");
    scanf("%d", &selected);
    switch (selected) {
    case 1:
        printf("How many rows whould you like to delete? ");
        scanf("%d", &newrows);
        printf("How many columns whould you like to add to the right side?
");
        scanf("%d", &newcols);
        newcols += cols;
        newrows = cols - newrows;
        result = allocMatrix(newrows, newcols);
        for (i = 0; i < newrows; i++) {
            for (j = 0; j < newcols; j++) {
                result[i][j] = mat[i][j];
            }
        }
        for (i = 0; i < cols; i++) {
            if ((i + 1) % (newrows) == 0) {
                db++;
            }
        }

        for (i = cols; i < newcols; i++) {
            for (j = 0; j < newrows; j++) {
                if ((((i + 1) - (newrows * db)) / (j + 1)) == 1 && (((i +
1)
                        - (newrows * db)) % (j + 1)) == 0) {
                    result[j][i] = 1;
                } else {
                    result[j][i] = 0;
                }
            }
            if ((i + 1) % (newrows) == 0) {
                db++;
            }
        }
        break;
        break;
    case 2:
        printf("How many rows whould you like to delete? ");
        scanf("%d", &newrows);
        newcols += cols;
        newrows = rows - newrows;
        result = allocMatrix(newrows, newcols);
        for (i = 0; i < newrows; i++) {
            for (j = 0; j < newcols; j++) {
                result[i][j] = mat[i][j];
            }
        }
        break;
    case 3:
```

```
        printf("How many columns whould you like to add to the right side?
");
        scanf("%d", &newcols);
        newcols += cols;
        newrows += rows;
        result = allocMatrix(newrows, newcols);
        for (i = 0; i < rows; i++) {
            for (j = 0; j < cols; j++) {
                result[i][j] = mat[i][j];
            }
        }
        for (i = 0; i < cols; i++) {
            if ((i + 1) % (newrows) == 0) {
                db++;
            }
        }

        for (i = cols; i < newcols; i++) {
            for (j = 0; j < newrows; j++) {
                if ((((i + 1) - (newrows * db)) / (j + 1)) == 1 && (((i +
1)
                        - (newrows * db)) % (j + 1)) == 0) {
                    result[j][i] = 1;
                } else {
                    result[j][i] = 0;
                }
            }
            if ((i + 1) % (newrows) == 0) {
                db++;
            }
        }
        break;
    }
    printf("matrix A =\n");
    printMatrix(mat, rows, cols);
    printf("new matrix A =\n");
    printMatrix(result, newrows, newcols);
    freeMatrix(result, newrows);
}

int main() {
    int rowA, colA;
    double **matA = NULL;

    printf("Number of rows of matrix A: ");
    scanf("%d", &rowA);
    printf("Number of columns of matrix A: ");
    scanf("%d", &colA);

    matA = allocMatrix(rowA, colA);
    randMatrix(matA, rowA, colA, 12, 16);

    cutMatrix(matA, rowA, colA);

    freeMatrix(matA, rowA);
    matA = NULL;
    return 0;
}
```

3.18.1.

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

const int limit=5;
typedef struct {
    FILE* bf;
    double* array;
    long int size;
} Store;

void initStore(Store* myStore) {
    printf("Size of the array: ");
    scanf("%ld", &myStore->size);
    if (myStore->size>limit)
        myStore->bf = fopen("temp.txt", "w+b");
    else
        myStore->array = (double*)malloc(sizeof(double)*myStore->size);
    return;
}

void delStore(Store* myStore) {
    if (myStore->size>limit) {
        fclose(myStore->bf); myStore->bf = NULL;
    } else
        free(myStore->array); myStore->array = NULL;
    return;
}

void readStore(Store* myStore) {
    double result;
    int idx;
    printf("Index of element to read: ");
    scanf("%d", &idx);
    if (myStore->size>limit) {
        fseek(myStore->bf, sizeof(double)*idx, SEEK_SET);
        fread(&result, sizeof(double), 1, myStore->bf);
    } else {
        result = myStore->array[idx];
    }
    printf("At %d there is: %lf\n", idx, result);
}

void writeStore(Store* myStore) {
    double value;
    int idx;
    printf("Index of element to write: ");
    scanf("%d", &idx);
    printf("value: ");
    scanf("%lf", &value);
    if (myStore->size>limit) {
        fseek(myStore->bf, sizeof(double)*idx, SEEK_SET);
        fwrite(&value, sizeof(double), 1, myStore->bf);
    //  fflush(myStore->bf);
    } else {
        myStore->array[idx] = value;
    }
```

```
    printf("%lf is written at %d\n", value, idx);
}

int main() {
    Store myStore={NULL, NULL, 0};
    int selection=0;

    initStore(&myStore);
    while (selection!=3) {
        printf("\nRead - 1\nWrite - 2\nQuit - 3\n");
        scanf("%d", &selection);
        switch (selection) {
            case 1:
                readStore(&myStore);
                break;
            case 2:
                writeStore(&myStore);
                break;
        }
    }
    delStore(&myStore);
    return 0;
}
```

3.19.1.

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

int main() {
    char words[][50]={"element", "size", "love", "ball", "toy", "gambit",
"cruiser",
        "shadow", "console", "Atlantis", "port", "staple", "Leonardo",
"bug",
        "hope", "vanity", "time", "issue", "fan", "strength"};
    char temp[50];
    int size=sizeof(words)/sizeof(*words); // sizeof(*words) = 50
    int trial, idxI, randIndex, errors=0;
    clock_t start, finish;
    double duration;

    srand((unsigned)time(NULL));
    printf("How much word would you like to type: ");
    scanf("%d", &trial);
    start = clock();
    for (idxI=0; idxI<trial; idxI++) {
        randIndex = rand() % size;
        printf("type: %s\n", words[randIndex]);
        scanf("%s", temp);
        while (strcmp(temp, words[randIndex]) != 0) {
            printf("error!\n");
            errors++;
            scanf("%s", temp);
        }
    }
    finish = clock();
    duration = (double)(finish - start) / CLOCKS_PER_SEC;
    printf("Ellapsed time: %4.2lf\n", duration);
```

```
    printf("Number of good trials: %d, bad trials: %d", trial, errors);
    return 0;
}
```

3.20.1.

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

#define INPUT_FILE "numbers.txt"
#define MAX_NUMBER_LENGTH 101
#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) > (b) ? (b) : (a))


char AddChar(char A, char B, char C)
{
  int n = (A - '0') + (B - '0') + (C - '0');
  return ( (n  % 10)) + '0';
}

char AddM(char A, char B, char C)
{
  int n = (A - '0') + (B - '0') + (C - '0');
  return ( (n / 10)) + '0';
}

void Add(char * NA, char * NB, char * C)
{
  char m;
  int AI, BI;
  char * A;
  char * B;
  int i, j;
  AI = strlen(NA);
  BI = strlen(NB);

  A = (AI > BI) ? NA : NB;
  B = (AI <= BI) ? NA : NB;
  if (BI > AI)
  {
    int temp = BI;
    BI = AI;
    AI = temp;
  }
  C[0] = '0';
  strcpy(C + 1, A);
  j = BI - 1;
  m = '0';
  for (i = AI - 1; i >= 0; i--)
  {
    if (j >= 0)
    {
      C[i + 1] = AddChar(A[i], B[j], m);
      m = AddM(A[i], B[j], m);
      j--;
    } else
    {
      if (m != '0')
```

```
      {
        C[i + 1] = AddChar(A[i], '0', m);
        m = AddM(A[i], '0', m);
      }
    }
  }
  C[0] = m;
  if (C[0] == '0')
    strcpy(C, C + 1);
}

void Read(FILE * fd)
{
  char Num1[MAX_NUMBER_LENGTH];
  char Num2[MAX_NUMBER_LENGTH];
  char Num3[MAX_NUMBER_LENGTH];
  fscanf(fd, "%s %s", Num1, Num2);
  printf("%s + %s = ", Num1, Num2);
  Add(Num1, Num2, Num3);
  printf("%s\n", Num3);
}

int main()
{
  FILE * fd = fopen(INPUT_FILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  Read(fd);
  return 0;
}
```

3.21.1.

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

#define DEFAULT_INPUT "datas.txt"

int Read(FILE * fd, int ** nums)
{
  int count, i;
  fscanf(fd, "%d", &count);
  printf("%d numbers:\n", count);
  *nums = (int *)malloc(sizeof(int) * count);
  for (i = 0; i < count; i++)
  {
    fscanf(fd, "%d", (*nums) + i);
    printf("%d ", (*nums)[i]);
  }
  printf("\n");
  return count;
}

double Median(int count, int * A)
{
  int i, j, pos, temp;
  for (i = 0; i < count - 1; i++)
```

```
  {
    pos = i;
    for (j = i; j < count; j++)
    {
      if (A[j] < A[pos])
        pos = j;
    }
    temp = A[i];
    A[i] = A[pos];
    A[pos] = temp;
  //  printf("%d ", A[i]);
  }

 // printf("%d\n", A[count - 1]);
  return count % 2 ? A[count / 2] : (A[ count / 2 - 1] + A[ count / 2]) /
2.0 ;
}

double Average(int count, int * A)
{
  long int sum = 0;
  int i;
  for (i = 0; i < count; i++)
    sum += A[i];
  return (double)sum / count;
}

int main(int argc, char * argv[])
{
  int count;
  int * datas = NULL;
  FILE * fd = fopen( argc > 0 ? argv[1] : DEFAULT_INPUT, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  count = Read(fd, &datas);
  fclose(fd);
  printf("The average is %g\n", Average(count, datas));
  printf("The median is: %g\n", Median(count, datas));
  free(datas);
  return 0;
}
```

3.22.1.

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

#define INPUT_FILE "students.txt"
#define MIN_AVERAGE 2.0
#define MIDDLE_AVERAGE 3.0
#define GOOD_AVERAGE 4.0
#define EXCELLENT_AVERAGE 4.5

#define MIN_BURSARY 15000
#define MIDDLE_BURSARY 20000
#define GOOD_BURSARY 25000
#define EXCELLENT_BURSARY 30000
```

```
struct TSubject
{
  int credit;
  int mark;
};

struct TStudent
{
  char Neptun[7];
  int SubjNum;
  struct TSubject * Subjects;
};

void ReadStudent(FILE * fd, struct TStudent * S)
{
  int i;
  fscanf(fd, "%s %d", S->Neptun, &S->SubjNum);
  printf("Neptun: %s %d subjects\n", S->Neptun, S->SubjNum);
  S->Subjects = (struct TSubject *)malloc(sizeof(struct TSubject) * S-
>SubjNum);
  for (i = 0; i < S->SubjNum; i++)
    fscanf(fd, "%d %d", &(S->Subjects + i)->credit, &(S->Subjects + i)-
>mark);
}

void Calc(struct TStudent * S, int SNum)
{
  int i, j, sum;
  int Credits, MaxCredits, Bursary;
  double Average;
  struct TSubject * Subj;
  for (i = 0; i < SNum; i++)
  {

    Subj = S->Subjects;
    Credits = MaxCredits = 0;
    sum = 0;
    for (j = 0; j < S->SubjNum; j++)
    {
      sum += Subj->credit * Subj->mark;
      MaxCredits += Subj->credit;
      if (Subj->mark >= 2)
        Credits += Subj->credit;
      Subj++;
    }
    Average = (double)sum / MaxCredits;
    if (Average < MIN_AVERAGE)
      Bursary = 0;
    else if (Average < MIDDLE_AVERAGE)
        Bursary = MIN_BURSARY;
    else if (Average < GOOD_AVERAGE)
        Bursary = MIDDLE_BURSARY;
    else if (Average < EXCELLENT_AVERAGE)
        Bursary = GOOD_BURSARY;
    else Bursary = EXCELLENT_BURSARY;
    printf("%s:\n\tCredits: %d/%d\n\tAverage: %g\n\tBursary: %d HUF\n",S-
>Neptun, MaxCredits, Credits, Average, Bursary);
    S++;
```

```
  }
}

int Read(FILE * fd, struct TStudent ** S)
{
  int num, i;
  fscanf(fd, "%d", &num);
  *S = (struct TStudent *)malloc(sizeof(struct TStudent) * num);
  for (i = 0; i < num; i++)
    ReadStudent(fd, (*S) + i);
  return num;
}

void FreeStudents(struct TStudent * S, int SNum)
{
  int i;
  for (i = 0; i < SNum; i++)
  {
    free(S[i].Subjects);
    S[i].Subjects = NULL;
  }
}

int main()
{
  int StudentNum;
  struct TStudent * Students;
  FILE * fd = fopen(INPUT_FILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  StudentNum = Read(fd, &Students);
  fclose(fd);
  Calc(Students, StudentNum);
  FreeStudents(Students, StudentNum);
  return 0;
}
```

3.23.1.

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

#define MAX_LENGTH 100
#define DEFAULT_INPUTFILE "dune.txt"
#define DEFAULT_WORD "Atreides"
#define FALSE 0
#define TRUE 1

int CheckWord(char * W, int len)
{
  int l = strlen(W);
  int i;
  for (i = len; i < l; i++)
  {
    if (((W[i] >= 'A') && (W[i] <= 'Z')) || ((W[i] >= 'a') && (W[i] <=
'z')))
```

```
      return FALSE;
  }
  return TRUE;
}

int SearchWord(FILE * fd, char * W)
{
  int count = 0;
  int len = strlen(W);
  char ReadedWord[MAX_LENGTH]={0};
  char BeforeWord[MAX_LENGTH]={0};
  BeforeWord[0] = 0;
  do
  {
    strcpy(BeforeWord, ReadedWord);
    ReadedWord[0] = 0;
    fscanf(fd, "%s", ReadedWord);
    if (strncmp(ReadedWord, W, len) == 0)
    {
      if (CheckWord(ReadedWord, len))
      {
        printf("%s %s\n",BeforeWord, ReadedWord);
        count++;
      }
    }
  } while (!feof(fd));
  return count;
}

int main(int argc, char *argv[])
{
  char * FileName = argc > 1 ? argv[1] : DEFAULT_INPUTFILE;
  char Word[MAX_LENGTH];
  FILE * fd = fopen(FileName, "r");
  if (fd == NULL)
  {
    perror(FileName);
    return 0;
  }
  if (argc > 2)
    strcpy(Word, argv[2]);
  else
    strcpy(Word, DEFAULT_WORD);
  printf("%d hits\n", SearchWord(fd, Word));
  fclose(fd);
  return 0;
}
```

3.24.1.

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

#define DEFAULT_INPUTFILE "script.txt"
#define MAX_COMMAND_LEN 10
#define MAX_OPERAND_LEN 5

#define STOP_CMD "STOP"
#define GET_CMD "GET"
```

```
#define ADD_CMD "ADD"
#define SUB_CMD "SUB"
#define MUL_CMD "MUL"
#define DIV_CMD "DIV"
#define WRITE_CMD "WRITE"

void Get(FILE * fd, int * N)
{
  char Op[MAX_OPERAND_LEN];
  fscanf(fd, "%s", Op);
  scanf("%d", N + (Op[0] - 'A'));
}

void WriteComment(FILE * fd)
{
  char ch;
  fscanf(fd, "%c", &ch);
  do
  {
    fscanf(fd, "%c", &ch);
    if (ch != '#')
      printf("%c", ch);
  } while (ch != '#');
}

void Add(FILE * fd, int * N)
{
  char Op1[MAX_OPERAND_LEN];
  char Op2[MAX_OPERAND_LEN];
  char ch1, ch2;
  fscanf(fd, "%s %s", Op1, Op2);
  ch1 = Op1[0]; ch2 = Op2[0];
  N[ ch1 - 'A' ] += N[ ch2 - 'A' ];
}

void Sub(FILE * fd, int * N)
{
  char Op1[MAX_OPERAND_LEN];
  char Op2[MAX_OPERAND_LEN];
  char ch1, ch2;
  fscanf(fd, "%s %s", Op1, Op2);
  ch1 = Op1[0]; ch2 = Op2[0];
  N[ ch1 - 'A' ] -= N[ ch2 - 'A' ];
}

void Mul(FILE * fd, int * N)
{
  char Op1[MAX_OPERAND_LEN];
  char Op2[MAX_OPERAND_LEN];
  char ch1, ch2;
  fscanf(fd, "%s %s", Op1, Op2);
  ch1 = Op1[0]; ch2 = Op2[0];
  N[ ch1 - 'A' ] *= N[ ch2 - 'A' ];
}

void Div(FILE * fd, int * N)
{
  char Op1[MAX_OPERAND_LEN];
  char Op2[MAX_OPERAND_LEN];
```

```
  char ch1, ch2;
  fscanf(fd, "%s %s", Op1, Op2);
  ch1 = Op1[0]; ch2 = Op2[0];
  N[ ch1 - 'A' ] /= N[ ch2 - 'A' ];
}

void Write(FILE * fd, int * N)
{
  char Op[MAX_OPERAND_LEN];
  fscanf(fd, "%s", Op);
  printf("%d\n", N[ Op[0] - 'A' ]);
}

void Run(FILE * fd)
{
  int Numbers[3];
  Numbers[0] = Numbers[1] = Numbers[2] = 0;
  char Command[MAX_COMMAND_LEN];
  do
  {
    fscanf(fd, "%s", Command);
    if (strcmp(Command, "#") == 0)
      WriteComment(fd);

    if (strcmp(Command, GET_CMD) == 0)
      Get(fd, Numbers);

    if (strcmp(Command, ADD_CMD) == 0)
      Add(fd, Numbers);

    if (strcmp(Command, SUB_CMD) == 0)
      Sub(fd, Numbers);

    if (strcmp(Command, MUL_CMD) == 0)
      Mul(fd, Numbers);

    if (strcmp(Command, DIV_CMD) == 0)
      Div(fd, Numbers);

    if (strcmp(Command, WRITE_CMD) == 0)
      Write(fd, Numbers);
  } while (strcmp(Command, STOP_CMD) != 0);
}

int main(int argv, char * argc[])
{
  FILE * fd = fopen(argv > 1 ? argc[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  Run(fd);
  fclose(fd);
  return 0;
}
```

3.25.1.

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

```
#include <stdlib.h>
#include <math.h>

#define DEFAULT_A 0
#define DEFAULT_B 1
#define DEFAULT_NUM 1000000

#define min(a, b) ((a) > (b)) ? ( b ) : ( a )
#define max(a, b) ((a) > (b)) ? ( a ) : ( b )

double f1(double x)
{
  return  sqrt( (2.0 - x) * x  ) * 4.0;
}

double f2(double x)
{
  return x * x;
}

void Integral(int a, int b, int num, double (*fptr)(double), char * funct
)
{
  int i;
  double res = 0.0;
  double interval = b - a;
  double mini = interval / num;
  double t;
  for (i = 0; i < num - 1; i++)
  {
    t = min(fptr(mini * i), fptr(mini * (i + 1))) * mini;
    res += t;
  }

  printf("The Riemann integral of %s over x from %d to %d: %g\n", funct,
a, b, res);
}

int main(int argc, char * argv[])
{
  int a, b, num;
  a = argc > 1 ? atoi(argv[1]) : DEFAULT_A;
  b = argc > 2 ? atoi(argv[2]) : DEFAULT_B;
  num = argc > 3 ? atoi(argv[3]) : DEFAULT_NUM;
  printf("\n");
  Integral(a, b, num, f1, "f1(x) ");
  Integral(a, b, num, f2, "f2(x) ");
  Integral(a, b, num, sin, "sin(x) ");
  Integral(a, b, num, tan, "tan(x) ");
  printf("\n");
  return 0;
}
```

3.26.1.

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

#define DEFAULT_INPUTFILE "polynoms.txt"
#define TRUE 1
```

```
#define FALSE 0

struct TPolynom
{
  int Degree;
  int * A;
};

void InitPolynom(struct TPolynom * P)
{
  P->Degree = 0;
  P->A = NULL;
}

void FreePolynom(struct TPolynom * P)
{
   free(P->A);
   P->A = NULL;
   P->Degree = 0;
}

void ReadPolynom(FILE * fd, struct TPolynom * P)
{
  int i;
  fscanf(fd, "%d", &(P->Degree));
  printf("Degree: %d\n", P->Degree);
  P->A = (int *)malloc(sizeof(int) * (P->Degree + 1));
  for (i = 0; i <= P->Degree; i++)
      fscanf(fd, "%d", P->A + i);
}

void PrintPolynom(struct TPolynom * P)
{
  int first = TRUE;
  int i;
  for (i = 0; i < P->Degree; i++)
  {
      if (P->A[i] != 0)
      {
         if (!first)
         {
            first = TRUE;
            if (P->A[i] > 0)
               printf(" + ");
         }
         if (P->A[i] < 0)
            printf(" - ");
         first = FALSE;
         if (abs(P->A[i]) == 1)
            printf("x");
         else
             printf("%dx", abs(P->A[i]));
         if (i < P->Degree - 1)
            printf("^%d", P->Degree - i);
      }
  }
  if (P->A[ P->Degree ] != 0)
  {
     if (first)
```

```
     {
        printf("%d", P->A[ P->Degree ]);
     } else
     {
       if (P->A[ P->Degree ] < 0)
          printf(" - %d", abs(P->A[ P->Degree ]));
       else
           printf(" + %d", abs(P->A[ P->Degree ]));
     }
  }
}

/* C = A + B */
void AddPolynom(struct TPolynom * A, struct TPolynom * B, struct TPolynom
* C)
{
     int i;
     if (C->A != NULL)
     {
        free(C->A);
        InitPolynom(C);
     }
     int * Max = A->Degree > B->Degree ? A->A : B->A;
     int * Min = A->Degree > B->Degree ? B->A : A->A;
     int min = A->Degree > B->Degree ? B->Degree : A->Degree;
     C->Degree = A->Degree > B->Degree ? A->Degree : B->Degree;
     C->A = (int*)malloc(sizeof(int) * (C->Degree + 1));
     for (i = 0; i <= C->Degree; i++)
            C->A[i] = Max[i];
     for (i = 0; i <= min; i++)
         C->A[i + ( C->Degree - min ) ] += Min[i];
}

int main(int argc, char *argv[])
{
  struct TPolynom A, B, C;
  InitPolynom(&A);
  InitPolynom(&B);
  InitPolynom(&C);
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
     perror("Error");
     return 0;
  }
  ReadPolynom(fd, &A);
  ReadPolynom(fd, &B);
  fclose(fd);
  PrintPolynom(&A);
  printf(" +\n");
  PrintPolynom(&B);
  printf(" =\n");
  AddPolynom(&A, &B, &C);
  PrintPolynom(&C);
  printf("\n");
  FreePolynom(&A);
  FreePolynom(&B);
  FreePolynom(&C);
  return 0;
```

```
}
```

3.27.1.

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

#define DEFAULT_INPUTFILE "codedmsg.txt"
#define WORD "THE"
#define MAX_WORD_LEN 15

typedef char TWord[MAX_WORD_LEN];

int ReadMessage(FILE * fd, TWord ** M)
{
  int num, i;
  fscanf(fd, "%d", &num);
  (*M) = (TWord*)malloc(sizeof(TWord) * num);
  for (i = 0; i < num; i++)
    fscanf(fd, "%s", (*M)[i]);
  return num;
}

void PrintMessage(TWord * M, int num)
{
  int i;
  for (i = 0; i < num; i++)
    printf("%s ", M[i]);
  printf("\n");
}

void Offset(TWord * M, int num, int offs)
{
  int i, j, len;
  for (i = 0; i < num; i++)
  {
    len = strlen(M[i]);
    for (j = 0; j < len; j++)
    {
      M[i][j] += offs;
      if (M[i][j] > 'Z')
        M[i][j] -= ('Z' - 'A' + 1 );
    }
  }
}

void Decoding(TWord * M, int num)
{
  int i = 0;
  int j = num;
  while ((i <= 'Z' - 'A') && (j >= num))
  {
    Offset(M, num, 1);
    j = 0;
    while ((j < num) && (strcmp(M[j], WORD) != 0))
      j++;
  }
}
```

```
int main(int argc, char * argv[])
{
  int num;
  TWord * Message;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  num = ReadMessage(fd, &Message);
  fclose(fd);
  printf("\nThe coded message: ");
  PrintMessage(Message, num);
  Decoding(Message, num);
  printf("The original message: ");
  PrintMessage(Message, num);
  printf("\n");
  free(Message);
  Message = NULL;
  return 0;
}
```

3.27.2.

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

#define DEFAULT_INPUTFILE "codedmsg.txt"
#define MAX_WORD_LEN 15

char WORD[500];

typedef char TWord[MAX_WORD_LEN];

int ReadMessage(FILE * fd, TWord ** M)
{
  int num, i;
  fscanf(fd, "%d", &num);
  (*M) = (TWord*)malloc(sizeof(TWord) * num);
  for (i = 0; i < num; i++)
    fscanf(fd, "%s", (*M)[i]);
  return num;
}

void PrintMessage(TWord * M, int num)
{
  int i;
  for (i = 0; i < num; i++)
    printf("%s ", M[i]);
  printf("\n");
}

void Offset(TWord * M, int num, int offs)
{
  int i, j, len;
  for (i = 0; i < num; i++)
  {
    len = strlen(M[i]);
```

```
    for (j = 0; j < len; j++)
    {
      M[i][j] += offs;
      if (M[i][j] > 'Z')
        M[i][j] -= ('Z' - 'A' + 1 );
    }
  }
}

void Decoding(TWord * M, int num)
{
  int i = 0;
  int j = num;
  while ((i <= 'Z' - 'A') && (j >= num))
  {
    Offset(M, num, 1);
    j = 0;
    while ((j < num) && (strcmp(M[j], WORD) != 0))
      j++;
  }
}

int main(int argc, char * argv[])
{
  int num;
  TWord * Message;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  num = ReadMessage(fd, &Message);
  fclose(fd);
  printf("\nKnown word: ");
  scanf("%s", WORD);
  printf("\nThe coded message: ");
  PrintMessage(Message, num);
  Decoding(Message, num);
  printf("The original message: ");
  PrintMessage(Message, num);
  printf("\n");
  free(Message);
  Message = NULL;
  return 0;
}

// codedmsg.txt
WKH TXLFN IRA
```

3.28.1.

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

#define DEFAULT_INPUTFILE "cds.txt"
#define MAX_WORD_LENGTH 16
#define TRUE 1
#define FALSE 0
```

```
typedef char TWord[MAX_WORD_LENGTH];

struct TCD
{
  int Num;
  TWord Name;
  TWord * P;
};

void ReadCD(FILE * fd, struct TCD * A)
{
  int i;
  fscanf(fd, "%s", A->Name);
  fscanf(fd, "%d", &A->Num);
  printf("%s:\n", A->Name);
  A->P = (TWord *)malloc(sizeof(TWord) * A->Num);
  for (i = 0; i < A->Num; i++)
  {
    fscanf(fd, "%s", A->P[i]);
    printf("\t%s\n", A->P[i]);
  }
}

int ReadCDs(FILE * fd, struct TCD ** A)
{
  int num, i;
  fscanf(fd, "%d", &num);
  (*A) = (struct TCD * )malloc(sizeof(struct TCD) * num);
  for (i = 0; i < num; i++)
    ReadCD(fd, (*A) + i);
  return num;
}

void Search(struct TCD * A, char * S, int num)
{
  int i = 0;
  int j;
  int success = FALSE;
  while ((i < num) && (!success))
  {
    j = 0;
    while ((j < A[i].Num) && (strcmp(A[i].P[j], S) != 0))
      j++;
    success = j < A[i].Num;
    i++;
  }
  if (success)
    printf("The %s is here: %s\n", S, A[i - 1].Name);
  else
    printf("%s does not exists!\n", S);
}

void FreeCDs(struct TCD * A, int num)
{
  int i;
  for (i = 0; i < num; i++)
  {
    free(A[i].P);
```

```
    A[i].P = NULL;
  }
}

int main(int argc, char * argv[])
{
  int CDNum;
  struct TCD * CDs;
  TWord Word;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  CDNum = ReadCDs(fd, &CDs);
  fclose(fd);
  printf("Program name: ");
  scanf(„%s”, Word);
  Search(CDs, Word, CDNum);
  FreeCDs(CDs, CDNum);
  free(CDs);
  CDs = NULL;
  return 0;
}
```

3.28.2.

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

#define DEFAULT_INPUTFILE "cds.txt"
#define MAX_WORD_LENGTH 16
#define TRUE 1
#define FALSE 0

typedef char TWord[MAX_WORD_LENGTH];

struct TCD
{
  int Num;
  TWord Name;
  TWord * P;
};

void ReadCD(FILE * fd, struct TCD * A)
{
  int i;
  fscanf(fd, "%s", A->Name);
  fscanf(fd, "%d", &A->Num);
  printf("%s:\n", A->Name);
  A->P = (TWord *)malloc(sizeof(TWord) * A->Num);
  for (i = 0; i < A->Num; i++)
  {
    fscanf(fd, "%s", A->P[i]);
    printf("\t%s\n", A->P[i]);
  }
}
```

```
int ReadCDs(FILE * fd, struct TCD ** A)
{
  int num, i;
  fscanf(fd, "%d", &num);
  (*A) = (struct TCD * )malloc(sizeof(struct TCD) * num);
  for (i = 0; i < num; i++)
    ReadCD(fd, (*A) + i);
  return num;
}

int spec_strcmp(const char* s1, const char* s2) {
  int l1=strlen(s1), l2=strlen(s2);
  int l=l1<l2?l1:l2,i;
  for (i=0;i<l;i++) {
    if (s1[i]!='?' && s1[i]!=s2[i] && s1[i]!='*') {
      return -1;
    }
  }
  return 0;
}

void Search(struct TCD * A, char * S, int num)
{
  int i = 0;
  int j;
  int success = FALSE;
  while ((i < num) && (!success))
  {
    j = 0;
    while ((j < A[i].Num) && (spec_strcmp(A[i].P[j], S) != 0))
      j++;
    success = j < A[i].Num;
    i++;
  }
  if (success)
    printf("The %s is here: %s\n", S, A[i - 1].Name);
  else
    printf("%s does not exists!\n", S);
}

void FreeCDs(struct TCD * A, int num)
{
  int i;
  for (i = 0; i < num; i++)
  {
    free(A[i].P);
    A[i].P = NULL;
  }
}

int main(int argc, char * argv[])
{
  int CDNum;
  struct TCD * CDs;
  TWord Word;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
```

```
    return 0;
  }
  CDNum = ReadCDs(fd, &CDs);
  fclose(fd);
  printf("Program name: ");
  scanf("%s", Word);
  Search(CDs, Word, CDNum);
  FreeCDs(CDs, CDNum);
  free(CDs);
  CDs = NULL;
  return 0;
}

// cds.txs
2
2007/11
2
BurningStudio
RadioRama
2005/4
3
Doc2PDF
Apollo
Stellarium
```

3.29.1.

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

#define DEFAULT_INPUTFILE "inventory.txt"
#define MAX_LENGTH 15
#define START_MONEY 10000

struct TGood
{
  int count;
  int purchaseprice;
  int shopprice;
  char name[MAX_LENGTH];
};

int ReadInventory(FILE * fd, struct TGood ** Good)
{
    int num, i;
    fscanf(fd, "%d", &num);
    (*Good) = (struct TGood*)malloc(sizeof(struct TGood) * num);
    for (i = 0; i < num; i++)
        fscanf(fd, "%s %d %d %d", (*Good)[i].name, &(*Good)[i].count,
                   &(*Good)[i].purchaseprice, &(*Good)[i].shopprice);
    return num;
}

void PrintInventory(struct TGood * G, int num)
{
     int i;
     printf("Inventory:\n************************************\n");
     for (i = 0; i < num; i++)
         printf("%s:\n\tCount: %d\n\tPurchase price: %d\n\tShop price:
%d\n",
```

```
            G[i].name, G[i].count, G[i].purchaseprice, G[i].shopprice);
}

int ReadLog(FILE * fd, int Money, int num, struct TGood * G)
{
     int lognum, i;
     int good, a;
     fscanf(fd, "%d", &lognum);
     for (i = 0; i < lognum; i++)
     {
         fscanf(fd, "%d %d", &good, &a);
         G[good - 1].count += a;
         if (a > 0) /* We buy the good */
            Money -= abs(a) * G[good - 1].purchaseprice;
         else /* We sell the good */
            Money += abs(a) * G[good - 1].shopprice;
     }
     return Money;
}

int main(int argc, char *argv[])
{
  struct TGood * Inventory;
  int GoodNum;
  int Money = START_MONEY;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
     perror("Error");
     return 0;
  }
  GoodNum = ReadInventory(fd, &Inventory);
  PrintInventory(Inventory, GoodNum);
  Money = ReadLog(fd, Money, GoodNum, Inventory);
  PrintInventory(Inventory, GoodNum);
  printf("We have got %d HUF\n", Money);
  fclose(fd);
  free(Inventory);
  Inventory = NULL;
  return 0;
}
```

3.30.1.

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

#define DEFAULT_INPUTFILE "lib1.txt"
#define MAXLEN 21
#define MAXUDC 4

// UDC = Universal Decimal Classification, in Hungarian: ETO

char * UDCClasses[10] = { "Generalities", "Philosophy, Psychology",
                          "Religion, Theology", "Social sciences",
                          "vacant", "Mathematics and natural sciences",
                          "Applied sciences, Medicine, Technology",
                          "Arts, Recreation, Entertainment, Sport" ,
                          "Language, Linguistics, Literature",
                          "Geography, Biography, History"};
```

```
struct TBook
{
  char title[MAXLEN];
  char author[MAXLEN];
  int edition;
  char UDC[MAXUDC];
};

int ReadBooks(FILE * fd, struct TBook ** Books)
{
  int num, i;
  struct TBook * B;
  fscanf(fd, "%d", &num);
  (*Books) = B = (struct TBook*)malloc(sizeof(struct TBook) * num);
  for (i = 0; i < num; i++)
  {
    fscanf(fd, "%s %s %d %s", B->title, B->author, &B->edition, B->UDC);
    B++;
  }
  return num;
}

void CountBooks(struct TBook * B, int num)
{
  int i;
  int types[10];
  for (i = 0; i < 10; i++)
    types[i] = 0;
  printf("\n\n");
  for (i = 0; i < num; i++)
  {
    printf("Author: %s\n Title: %s\nYear of edition: %d\nUDC: %s %s\n\n",
B[i]. author, B[i].title, B[i].edition,
    B[i].UDC, UDCClasses[ B[i].UDC[0] - '0' ]);
    types[ B[i].UDC[0] - '0' ]++;
  }
  printf("\n************************************\n\n");
  for (i = 0; i < 10; i++)
    printf("%s: %d books\n", UDCClasses[i], types[i]);
  printf("\n\n");

}

int main(int argc, char * argv[])
{
  int booknum;
  struct TBook * Books;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  booknum = ReadBooks(fd, &Books);
  fclose(fd);
  CountBooks(Books, booknum);
  free(Books);
  Books = NULL;
```

```
  return 0;
}
```

3.31.1.

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

#define DEFAULT_INPUTFILE "lib1.txt"
#define MAXLEN 20
#define YEAR 2008
#define MONTH 12
#define DAY 15
#define TRUE 1
#define FALSE 0

struct TBook
{
  char title[MAXLEN];
  char author[MAXLEN];
    int year, month, day;
};

int ReadBooks(FILE * fd, struct TBook ** Books)
{
  int num, i;
  struct TBook * B;
  fscanf(fd, "%d", &num);
  (*Books) = B = (struct TBook*)malloc(sizeof(struct TBook) * num);
  for (i = 0; i < num; i++)
  {
    fscanf(fd, "%s %s %d %d %d", B->title, B->author, &B->year, &B-
>month, &B->day);
    B++;
  }
  return num;
}

int Later(int y, int m, int d)
{
  if (y < YEAR)
    return TRUE;
  else if (y == YEAR)
  {
    if (m < MONTH)
      return TRUE;
    else if (m == MONTH)
    {
      if (d < DAY)
        return TRUE;
    }
  }
  return FALSE;
}

void PrintBooks(struct TBook * B, int num)
{
  int i;
  printf("\n");
  for (i = 0; i < num; i++)
```

```
  {
    if (Later(B[i].year, B[i].month, B[i].day))
      printf("Title: %s\nAuthor: %s\nExpiraton: %d %d %d\n\n",
        B[i].title, B[i].author, B[i].year, B[i].month, B[i].day);
  }
}

int main(int argc, char * argv[])
{
  int booknum;
  struct TBook * Books;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  booknum = ReadBooks(fd, &Books);
  fclose(fd);
  PrintBooks(Books, booknum);
  free(Books);
  Books = NULL;
  return 0;
}
```

3.32.1.

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

#define MAXLEN 21
#define DEFAULT_INPUTFILE "dict.txt"

void Search(FILE * fd, char * word)
{
  int num, i, hits = 0;
  char hword[MAXLEN], eword[MAXLEN];
  fscanf(fd, "%d", &num);
  eword[0] = 0;
  for (i = 0; i < num; i++)
  {
    fscanf(fd, "%s %s", eword, hword);
    if (strncmp(eword, word, strlen(word)) == 0)
    {
      printf("Eng->Hun: %s %s\n", eword, hword);
      hits++;
    }
    if (strcmp(hword, word) == 0)
    {
      printf("Hun->Eng: %s %s\n", hword, eword);
      hits++;
    }
  }
  printf("\n%d hits\n\n", hits);
}

int main(int argc, char * argv[])
{
  char word[MAXLEN];
```

```
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  printf("\nType the word: ");
  scanf("%s", word);
  Search(fd, word);
  fclose(fd);
  printf("\n");
  return 0;
}
```

3.33.1.

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

#define TRUE 1
#define FALSE 0
#define DEFAULT_INPUTFILE "sudoku1.txt"

typedef int TTable[9][9];

void ReadTable(FILE * fd, TTable T)
{
  int i, j;
  for (i = 0; i < 9; i++)
  {
    for (j = 0; j < 9; j++)
      fscanf(fd, "%d", &T[i][j]);
  }
}

void PrintTable(TTable T)
{
  int i, j;
    for (i = 0; i < 9; i++)
    {
        for (j = 0; j < 9; j++)
            printf("%d ", T[i][j]);
        printf("\n");
    }
}

int CheckRowsColumns(TTable T)
{
    int i, j, n, num;
    for (i = 0; i < 9; i++)
    {
        for (j = 0; j < 9; j++)
        {
            n = i ? 0 : 1;
            num = T[i][j];
            while ((n < 9) && (T[n][j] != num))
                n += (i == n + 1) ? 2 : 1;
            if (n < 9)
            {
                printf("This is a wrong table!\n");
```

```
                return FALSE;
            }

            n = j ? 0 : 1;
            num = T[i][j];
            while ((n < 9) && (T[i][n] != num))
                n += (j == n + 1) ? 2 : 1;
            if (n < 9)
            {
                printf("This is a wrong table!\n");
                return FALSE;
            }
        }
    }
    return TRUE;
}

int CheckOneCell(TTable T, int x, int y)
{
    int i, j, num, err;
    for (num = 1; num <= 9; num++)
    {
        err = TRUE;
        for (i = y * 3; i < (y + 1) * 3; i++)
        {
            for (j = x * 3; j < (x + 1) * 3; j++)
            {
                if (T[i][j] == num)
                    err = FALSE;
            }
        }
        if (err)
        {
            printf("This is a wrong table!\n");
            return FALSE;
        }
    }
    return TRUE;
}

void CheckCells(TTable T)
{
    int i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            if (!CheckOneCell(T, i, j))
                return;
    printf("This table is correct!\n");
}

void Check(TTable T)
{
    if (CheckRowsColumns(T))
        CheckCells(T);
}

int main(int argc, char * argv[])
{
  TTable Table;
```

```
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  ReadTable(fd, Table);
  fclose(fd);
    PrintTable(Table);
    Check(Table);
  return 0;
}
```

3.34.1.

```
#include <stdio.h>

#define FIRST_PLAYER 0
#define SECOND_PLAYER 1
#define TRUE 1
#define FALSE 0
#define SIGN1 'X'
#define SIGN2 'O'
#define SIZE 3
#define NUMTOWIN 3

typedef char TTable[SIZE][SIZE];

void InitTable(TTable T)
{
  int i, j;
  for (i = 0; i < SIZE; i++)
    for (j = 0; j < SIZE; j++)
      T[i][j] = ' ';
}

void PrintTable(TTable T)
{
  int i, j;
  printf("   ");
  for (i = 'A'; i < 'A' + SIZE; i++)
    printf("%c ", i);
  printf("\n");
  printf("  -");
  for (i = 0; i < SIZE; i++)
    printf("--");
  printf("\n");
  for (i = 0; i < SIZE; i++)
  {
    printf("%2d|", i + 1);
    for (j = 0; j < SIZE; j++)
      printf("%c|", T[j][i]);
    printf("\n  ");
    for (j = 0; j < SIZE; j++)
      printf("--");
    printf("-\n");
  }

}
```

```
int GetY(char y1, char y2)
{
  return y2 > 0 ? (y1 - '0') * 10 + (y2 - '1') : y1 - '1' ;
}

int Drawn(TTable T)
{
  int i, j;
  for (i = 0; i < SIZE; i++)
      for (j = 0; j < SIZE; j++)
          if (T[i][j] == ' ')
             return FALSE;
  return TRUE;
}

int Win(TTable T, char ch)
{
    int i, j;
    for (i = 0; i < SIZE; i++)
    {
        for (j = 0; j < SIZE; j++)
        {
            if (i <= SIZE - NUMTOWIN )
            {
               if ((T[i][j] == ch) && (T[i+1][j] == ch) && (T[i+2][j] ==
ch))
                  return TRUE;
            }
            if (j <= SIZE - NUMTOWIN )
            {
               if ((T[i][j] == ch) && (T[i][j+1] == ch) && (T[i][j+2] ==
ch))
                  return TRUE;
            }
            if ((i <= SIZE - NUMTOWIN ) && (j <= SIZE - NUMTOWIN ))
            {
               if ((T[i][j] == ch) && (T[i+1][j+1] == ch) && (T[i+2][j+2]
== ch))
                  return TRUE;
            }
            if ((i >= NUMTOWIN - 1 ) && (j <= SIZE - NUMTOWIN ))
            {
               if ((T[i][j] == ch) && (T[i-1][j+1] == ch) && (T[i-2][j+2]
== ch))
                  return TRUE;
            }
        }
    }
    return FALSE;
}

int WrongCoord(TTable T, char x, char y1, char y2)
{
  int y = GetY(y1, y2);
  if ((x < 'A') || (x >= 'A' + SIZE ) || (y < 0) || (y >= SIZE))
  {
    printf("Wrong coordinate!\n");
    return TRUE;
  }
```

```
  if (T[x - 'A'][y] != ' ')
  {
    printf("You already cannot choose this position!\n");
    return TRUE;
  }
  return FALSE;
}



void Play(TTable T)
{
  char target[5];
  int player = FIRST_PLAYER;
  do
  {
    if (player == FIRST_PLAYER)
      printf("First player\n");
    else
      printf("Second player\n");
    printf("Target: ");
    do
    {
      scanf("%s", target);
    } while ((target[0] != '0') && WrongCoord(T, target[0], target[1],
target[2]));
    if (target[0] != '0')
    {
      if (player == FIRST_PLAYER)
      {
        T[target[0] - 'A'][GetY(target[1], target[2])] = SIGN1;
        if (Win(T, SIGN1))
        {
           PrintTable(T);
           printf("First player won!\n");
           return;
        }
        player = SECOND_PLAYER;
      } else
      {
        T[target[0] - 'A'][GetY(target[1], target[2])] = SIGN2;
        if (Win(T, SIGN2))
        {
           PrintTable(T);
           printf("Second player won!\n");
           return;
        }
        player = FIRST_PLAYER;
      }
      PrintTable(T);
    }
  } while ((target[0] != '0') && !Drawn(T));
  printf("Game over!\n");
}

int main()
{
  TTable Table;
  InitTable(Table);
```

```
  PrintTable(Table);
  Play(Table);
  return 0;
}
```

3.35.1.

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

#define DEFAULT_INPUTFILE "map1.txt"
#define HILL 250
#define MOUNTAIN 500
#define HIGHMOUNTAIN 1500

struct TMap
{
  int r, c;
  int ** m;
};

void ReadMap(FILE * fd, struct TMap * M)
{
  int i, j;
  fscanf(fd, "%d %d", &M->r, &M->c);
  M->m = (int**)malloc(sizeof(int*) * M->r);
  for (i = 0; i < M->r; i++)
  {
    M->m[i] = (int*)malloc(sizeof(int) * M->c);
    for (j = 0; j < M->c; j++)
      fscanf(fd, "%d", &M->m[i][j]);
  }
}

void FreeMap(struct TMap * M)
{
  int i;
  for (i = 0; i < M->r; i++)
  {
    free(M->m[i]);
    M->m[i] = NULL;
  }
  free(M->m);
  M->m = NULL;
}

void PrintMap(struct TMap * M)
{
  int i, j;
  printf("\nThe map:\n\n");
  for (i = 0; i < M->r; i++)
  {
    for (j = 0; j < M->c; j++)
      printf("%4d ", M->m[i][j]);
    printf("\n");
  }
}

void Count(struct TMap * M)
{
```

```
  int lowland = 0, hill = 0, mountain = 0, highmountain = 0;
  int i, j, h;
  double A = M->r * M->c;
  for (i = 0; i < M->r; i++)
  {
    for (j = 0; j < M->c; j++)
    {
      h = M->m[i][j];
      if (h < HILL)
        lowland++;
      else if (h < MOUNTAIN)
        hill++;
      else if (h < HIGHMOUNTAIN)
        mountain++;
      else highmountain++;
    }
  }
  printf("\nLowland: %g %%\n", lowland / A * 100.0);
  printf("Hill: %g %%\n", hill / A * 100.0);
  printf("Mountain: %g %%\n", mountain / A * 100.0);
  printf("High mountain: %g %%\n\n", highmountain / A * 100.0);
}

int main(int argc, char * argv[])
{
  struct TMap Map;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  ReadMap(fd, &Map);
  fclose(fd);
  PrintMap(&Map);
  Count(&Map);
  FreeMap(&Map);
  return 0;
}
```

3.36.1.

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

#define SIZE 3

typedef double Matrix[SIZE][SIZE];

void PrintMatrix(Matrix m)
{
  int i, j;
  printf("\n");
  for (i = 0; i < SIZE; i++)
  {
       for (j = 0; j < SIZE; j++)
      printf("%2g ", m[i][j]);
    printf("\n");
  }
```

```
  printf("\n");
}

double GetDeterminant(Matrix m)
{
  return m[0][0] * m[1][1] * m[2][2] +
         m[0][1] * m[1][2] * m[2][0] +
         m[0][2] * m[1][0] * m[2][1] -
         m[0][2] * m[1][1] * m[2][0] -
         m[0][1] * m[1][0] * m[2][2] -
         m[0][0] * m[1][2] * m[2][1];
}

void MakeAdjMatrix(Matrix m)
{
  int i, j, x1, y1, x2, y2;
  Matrix m2;
  for (i = 0; i < SIZE; i++)
  {
    for (j = 0; j < SIZE; j++)
    {
      x1 = j == 0; // if j = 0, x1 = 1, if j = 1, x1 = 0, if j = 2, x1 =
0
      y1 = i == 0;
      // if j = 2, x2 = 1, if j = 1, x2 = 2, if j = 0, x2 = 2
      x2 = (j != 2) + 1;
      y2 = (i != 2) + 1;
      m2[j][i] = (i + j) % 2 ? -1 : 1;
      m2[j][i] *= (m[y1][x1] * m[y2][x2] - m[y1][x2] * m[y2][x1]);
    }
  }
  memcpy(m, m2, sizeof(double) * SIZE * SIZE);
}

void InvertMatrix(Matrix m)
{
  int i, j;
  double det;
  det = GetDeterminant(m);
  printf("Adjugate matrix: \n");
  MakeAdjMatrix(m);
  PrintMatrix(m);

  printf("Determinant of the matrix: %g\n", det);
  if (det != 0.0)
  {
    for (i = 0; i < SIZE; i++)
    {
      for (j = 0; j < SIZE; j++)
        m[i][j] /= det;
    }
    printf("Inverse matrix: \n");
    PrintMatrix(m);
  } else printf("We cannot invert this matrix!\n");
}

int main()
{
  Matrix matrix= { {1, 2, 3},
```

```
                    {2, 4, 5},
                    {3, 5, 6} };
  printf("Original matrix:\n");
  PrintMatrix(matrix);
  InvertMatrix(matrix);
  return 0;
}
```

3.37.1.

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

#define DEFAULT_INPUTFILE "matrices1.txt"

struct TMatrix
{
  int r,c;
  int ** m;
};

void InitMatrix(struct TMatrix * M, int r, int c)
{
  int i,j;
  M->r = r;
  M->c = c;
  M->m = (int**)malloc(sizeof(int*) * r);
  for (i = 0; i < r; i++)
  {
    M->m[i] = (int*)malloc(sizeof(int) * c);
    for (j = 0; j < c; j++)
      M->m[i][j] = 0;
  }
}

void FreeMatrix(struct TMatrix * M)
{
  int i;
  for (i = 0; i < M->r; i++)
  {
    free(M->m[i]);
    M->m[i] = NULL;
  }
  free(M->m);
  M->m = NULL;
  M->r = M-> c = 0;
}

void PrintMatrix(struct TMatrix * M)
{
  int i, j;
  for (i = 0; i < M->r; i++)
  {
    for (j = 0; j < M->c; j++)
      printf("%3d ", M->m[i][j]);
    printf("\n\n\n");
  }
}

void ReadMatrix(FILE * fd, struct TMatrix * M)
```

```
{
  int i, j;
  for (i = 0; i < M->r; i++)
  {
    for (j = 0; j < M->c; j++)
      fscanf(fd, "%d", &M->m[i][j]);
  }
}

void AddMulMatrix(struct TMatrix * A, struct TMatrix * B, struct TMatrix
* C, int m)
{
  int i, j;
  for (i = 0; i < A->r; i++)
  {
    for (j = 0; j < A->c; j++)
      C->m[i][j] = (A->m[i][j] + B->m[i][j]) * m;
  }
}

int main(int argc, char *argv[])
{
  struct TMatrix A, B, C;
  int r, c;
  FILE * fd = fopen( argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  fscanf(fd, "%d %d", &r, &c);
  InitMatrix(&A, r, c);
  InitMatrix(&B, r, c);
  InitMatrix(&C, r, c);
  ReadMatrix(fd, &A);
  ReadMatrix(fd, &B);
  fclose(fd);
  printf("A : \n\n");
  PrintMatrix(&A);
  printf("B : \n\n");
  PrintMatrix(&B);
  AddMulMatrix(&A, &B, &C, 2);
  printf("************************\n (A + B) * 2 = \n\n");
  PrintMatrix(&C);
  FreeMatrix(&A);
  FreeMatrix(&B);
  FreeMatrix(&C);
  return 0;
}
```

3.38.1.

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

#define SZOKOZ "....."

char * morsecodes[26] = { ".=", "=...", "=.=.",    // A, B, C
                          "=..", ".", "..=.",      // D, E, F
```

```
                              "==.", "....", "..",     // G, H, I
                              ".===", "=.=", ".=..",   // J, K, L
                              "==", "=.", "===",      // M, N, O
                              ".==.", "==.=", ".=.",   // P, Q, R
                              "...", "=", "..=",      // S, T, U
                              "...=", ".==", "=..=",   // V, W, X
                              "=.==", "==.."};    // Y, Z

void getline(char * s, int maxlen)
{
  char ch;
  int i = 0;
  do
  {
    ch = getchar();
    if (ch!='\n')
      s[i++] = ch;
  }
  while ((i < maxlen - 1) && (ch!='\n'));
  s[i] = 0;
}

char * coding(char * s)
{
  int len = 1;
  int i;
  char * c;
  for (i = 0; i < strlen(s); i++)
    if (s[i] != ' ')
      len += strlen(morsecodes[ s[i] - 'A' ]);
    else
      len += strlen(SZOKOZ);
  c = (char*)malloc(sizeof(char) * len);
  //printf("len: %d\n", len);
  len = 0;
  for (i = 0; i < strlen(s); i++)
  {
    if (s[i] != ' ')
    {
      //printf("char: %c index: %d code: %s\n", s[i], s[i] - 'A',
morsecodes[ s[i] - 'A']);
      strcpy(c + len, morsecodes[ s[i] - 'A' ]);
      len += strlen(morsecodes[ s[i] - 'A' ]);
    } else
    {
      strcpy(c + len, SZOKOZ);
      len += strlen(SZOKOZ);
    }
  }
  c[len] = 0;
  return c;
}

int main()
{
  char message[100];
  char * morse;
  printf("The message: ");
  getline(message, 100);
```

```
  printf("Message: \"%s\" \n", message);
  morse = coding(message);
  printf("The coded message: \"%s\"\n", morse);
  free(morse);
  morse = NULL;
  return 0;
}
```

3.39.1

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

#define DEFAULT_INPUTFILE "input.txt"

struct TMatrix
{
  int r,c;
  int ** m;
};

void InitMatrix(struct TMatrix * M, int r, int c)
{
  int i,j;
  M->r = r;
  M->c = c;
  M->m = (int**)malloc(sizeof(int*) * r);
  for (i = 0; i < r; i++)
  {
    M->m[i] = (int*)malloc(sizeof(int) * c);
    for (j = 0; j < c; j++)
      M->m[i][j] = 0;
  }
}

void FreeMatrix(struct TMatrix * M)
{
  int i;
  for (i = 0; i < M->r; i++)
  {
    free(M->m[i]);
    M->m[i] = NULL;
  }
  free(M->m);
  M->m = NULL;
  M->r = M-> c = 0;
}

void PrintMatrix(struct TMatrix * M)
{
  int i, j;
  for (i = 0; i < M->r; i++)
  {
    for (j = 0; j < M->c; j++)
      printf("%3d ", M->m[i][j]);
    printf("\n\n\n");
  }
}

void ReadMatrix(FILE * fd, struct TMatrix * M)
```

```
{
  int i, j;
  for (i = 0; i < M->r; i++)
  {
    for (j = 0; j < M->c; j++)
      fscanf(fd, "%d", &M->m[i][j]);
  }
}

void AddMulMatrix(struct TMatrix * A, struct TMatrix * B, struct TMatrix
* C, int m)
{
  int i, j;
  for (i = 0; i < A->r; i++)
  {
    for (j = 0; j < A->c; j++)
      C->m[i][j] = (A->m[i][j] + B->m[i][j]) * m;
  }
}

void MulMatrixVector(struct TMatrix * M, struct TMatrix * V1, struct
TMatrix * V2)
{
  int i, j;
  for (i = 0; i < M->r; i++)
  {
    V2->m[i][0] = 0;
    for (j = 0; j < M->c; j++)
      V2->m[i][0] += M->m[i][j] * V1->m[j][0];
  }
}

int main(int argc, char *argv[])
{
  struct TMatrix A, V1, V2;
  int r, c;
  FILE * fd = fopen( argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  fscanf(fd, "%d %d", &r, &c);
  InitMatrix(&A, r, c);
  InitMatrix(&V1, c, 1);
  InitMatrix(&V2, r, 1);
  ReadMatrix(fd, &A);
  ReadMatrix(fd, &V1);
  fclose(fd);
  MulMatrixVector(&A, &V1, &V2);
  printf("Matrix : \n\n");
  PrintMatrix(&A);
  printf("Vector : \n\n");
  PrintMatrix(&V1);
  printf("**********************\n Matrix * Vector = \n\n");
  PrintMatrix(&V2);
  FreeMatrix(&A);
  FreeMatrix(&V1);
  FreeMatrix(&V2);
```

```
  return 0;
}
```

3.40.1.

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

#define MAXLEN 101

int Count(char * str)
{
  int szóközs = 0;
  int i;
  for (i = 0; str[i] != 0; i++)
    szóközs += (str[i] == ' ');
  return ++szóközs;
}

char * GetToken(char * s, int * len)
{
  int i = 0;
  char * w;
  *len = 0;
  while ((s[ *len ] != 0) && (s[ *len ] != ' '))
    (*len)++;

  (*len)++;
  w = (char*)malloc(sizeof(char) * (*len));
  for (i = 0; i < (*len) - 1; i++)
    w[i] = s[i];
  w[(*len) - 1] = 0;
  return w;
}

char ** Tokenizer(char * str, int * s)
{
  int i;
  int size = Count(str);
  int len;
  char ** t = (char **)malloc(size * sizeof(char*));
  for (i = 0; i < size; i++)
  {
    t[i] = GetToken(str, &len);
    str += len;
  }
  *s = size;
  return t;
}

void Print(char ** t, int size)
{
  int i;
  for (i = 0; i < size; i++)
    printf("%d.: \"%s\"\n", i + 1, t[i]);
}

void Free(char *** t, int size)
{
  int i;
```

```
  char ** t2 = *t;
  for (i = 0; i < size; i++)
  {
    free(t2[i]);
    t2[i] = NULL;
  }
  *t = NULL;
}

int main()
{
  char str[MAXLEN];
  char ** t = NULL;
  int size;
  if (gets(str) == NULL)
  {
    printf("Error!\n");
    return 0;
  }
  printf("The typed text: \"%s\"\n", str);
  t = Tokenizer(str, &size);
  Print(t, size);
  Free(&t, size);

  return 0;
}
```

3.41.1.

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

#define DEFAULT_INPUTFILE "text.txt"
#define MAX_TEXT_LENGTH 1001
#define MAX_WORD_LENGTH 21
#define TRUE 1
#define FALSE 0

void Change(char * t, char * w1, char * w2)
{
  int i = 0;
  int j;
  int lent = strlen(t);
  int lenw = strlen(w1);
  int lenw2 = strlen(w2);
  char tmp;
  int hits = 0;
  for (i = 0; i < lent - lenw; i++)
  {
    if (strncmp(t + i, w1, lenw) == 0)
    {
      tmp = t[i + lenw2];
      strcpy(t + i, w2);
      t[i + lenw2] = tmp;
      for (j = i + lenw2; j < lent - (lenw - lenw2); j++)
        t[j] = t[j + lenw - lenw2];
      t[j] = 0;
      lent = strlen(t);
      hits++;
```

```
    }
  }
  printf("\n%d hits\n", hits);
}

int main(int argc, char * argv[])
{
  char text[MAX_TEXT_LENGTH];
  char word1[MAX_WORD_LENGTH];
  char word2[MAX_WORD_LENGTH];
  FILE * fd = fopen(argc > 1? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  if (fgets(text, MAX_TEXT_LENGTH - 1, fd) == NULL)
  {
    perror("Error");
    fclose(fd);
    return 0;
  }
  fclose(fd);
  printf("\nThe original text: \"%s\"\n\n", text);
  printf("Type a word: ");
  scanf("%s", word1);
  printf("Type the new word: ");
  scanf("%s", word2);
  Change(text, word1, word2);
  printf("\nThe new text: \"%s\"\n\n", text);
  return 0;
}
```

3.42.1.

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

#define DEFAULT_INPUTFILE "message.txt"
#define TRUE 1
#define FALSE 0
#define MAXLEN 51

typedef unsigned char ubyte;

ubyte GetChecksum(char * msg)
{
  unsigned short sum = 0;
  int i;
  for (i = 0; msg[i] != 0; i++)
    sum += (unsigned char)msg[i];
  printf("\nSum of \"%s\": %X\n", msg, sum);
  // drop the carry
  sum = (ubyte)sum;
  // two's complement:
  sum = 0xFF - sum + 1;
  printf("The checksum: %X\n", sum);
  return sum;
}
```

```
int Checking(char * msg, ubyte chksm)
{
  int i;
  unsigned short sum = chksm;
  for (i = 0; msg[i] != 0; i++)
    sum += (unsigned char)msg[i];
  sum = (ubyte)sum;
  return (sum == 0);
}

void ReadMessages(FILE * fd)
{
  int count;
  int i;
  unsigned int chksum;
  char msg[MAXLEN];
  fscanf(fd, "%d\n", &count);
  printf("\n");
  for (i = 0; i < count; i++)
  {
    if (fgets(msg, MAXLEN, fd) == NULL)
    {
      printf("Error\n");
      return;
    }
    msg[strlen(msg) - 1] = 0;
    fscanf(fd, "%X\n", &chksum);
    printf("\"%s\"  %X ===> ", msg, chksum);
    if (Checking(msg, chksum))
      printf("Correct!\n");
    else
      printf("Faulty!\n");
  }
}

int main(int argc, char * argv[])
{
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 0;
  }
  ReadMessages(fd);
  fclose(fd);
  GetChecksum("This is a simple checksum example.");
  printf("\n");
  return 0;
}
```

3.43.1.-3.34.3.

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

#define AKTUALIS_EV 2010

int main(int argc, char** argv) {
```

```
    FILE * fd = fopen(argv[1], "r");
    if (fd == NULL) {
        perror("Hiba");
        return EXIT_FAILURE;
    }
    int kiskoru = 0;
    int felnott = 0;
    int nyugdijas = 0;
    int vh2 = 0;
    int szokoevben = 0;
    int eletkor;
    while (!feof(fd)) {

        if (fscanf(fd, "%d", &eletkor)) {
            // 4.30.1:
            if (eletkor < 18) {
                kiskoru++;
            } else {
                felnott++;
                if (eletkor >= 62) {
                    nyugdijas++;
                }
            }
            // 4.30.2:
            int ev = AKTUALIS_EV - eletkor;
            if (ev >= 1939 && ev <= 1945) {
                vh2++;
            }
            // 4.30.3:
            printf("ev: %d\n", ev);
            if ((ev % 400 == 0) || (ev % 4 == 0 && ev % 100 != 0)) {
                printf("szokoev\n");
                szokoevben++;
            }
        }
    }
    fclose(fd);

    printf("Kiskoruak: %d\n", kiskoru);
    printf("Felnottek: %d\n", felnott);
    printf("Nyugdijasok: %d\n", nyugdijas);
    printf("Vilaghaboru alatt: %d\n", vh2);
    printf("Szokoevben: %d\n", szokoevben);
    return EXIT_SUCCESS;
}

// input.txt
3
20
30
3
5
23
53
63
13
70
80
43
```

```
23
64
23
64
2
3
10
32
53
110
```

3.44.1.

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

int main() {
    int koteg;
    int osszeg = 0;
    do {
        printf("Kerem a keriteskoteg hosszat: ");
        scanf("%d", &koteg);
        osszeg += koteg;
    } while (koteg > 0);
    printf("Osszesen van %d meter kerites\n", osszeg);
    if (osszeg < 400) {
        printf("Meg kell %d meternyi kerites\n", 400 - osszeg);
    }
    return EXIT_SUCCESS;
}
```

3.45.1.

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

#define N 10

int main() {
    int i;
    int osszeg = 0;
    int jegy;
    for (i = 0; i < N; i++) {
        printf("%d. jegy: ", i + 1);
        scanf("%d", &jegy);
        osszeg += jegy;
    }
    printf("A jegyek szamtani atlaga: %g\n", (double)osszeg / N);
    return EXIT_SUCCESS;
}
```

3.45.2-3.45.3.

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

#define N 10

typedef struct lista {
    struct lista * kovetkezo;
    int jegy;
```

```
} lista;

int main() {
    lista * jegyek = NULL;
    int i;
    int osszeg = 0;
    int jegy;
    double atlag;
    i = 1;

    do {
        printf("%d. jegy: ", i++);
        scanf("%d", &jegy);
        osszeg += jegy;
        if (jegy > 0) {
            lista * elem = (lista *) malloc(sizeof (lista));
            elem->jegy = jegy;
            if (jegyek == NULL) {
                elem->kovetkezo = NULL;
            } else {
                elem->kovetkezo = jegyek;
            }
            jegyek = elem;
        }
    } while (jegy > 0);
    atlag = (double) osszeg / (i - 2);

    lista * temp = jegyek;
    lista * prev;
    double elteres = 0;
    while (temp != NULL) {
        elteres += (atlag - temp->jegy) * (atlag - temp->jegy);
        prev = temp;
        temp = temp->kovetkezo;
        free(prev);
    }
    double szoras = sqrt(elteres / (i - 2));

    printf("A jegyek szamtani atlaga: %g\n", atlag);
    printf("A jegyek szorasa: %g\n", szoras);
    return EXIT_SUCCESS;
}
```

3.46.1.-3.46.2.

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

int main() {
    unsigned long long int termekeny = 0, termeketlen = 1;
    unsigned long long int osszesen = termeketlen;
    int honap;
    int ev;
    for (ev = 1; ev <= 5; ev++) {
        for (honap = 1; honap <= 12; honap++) {
            unsigned long long int nyulakUj = termekeny + termeketlen;
            osszesen += nyulakUj;
            termekeny = termeketlen;
            termeketlen = nyulakUj;
            printf("%d\n", nyulakUj);
```

```
        }

        termekeny *= 0.1;
        termeketlen *= 0.05;
        printf("eladas utan %llu termekeny es %llu termeketlen nyulpar
van\n",
                termekeny, termeketlen);

        printf("Osszesen %llu adag tapra lesz szukseg\n", osszesen * 2);
    }
    return EXIT_SUCCESS;
}
```

3.47.1.

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

#define N 30

int main() {
    double jegyek[N];
    int i;
    for (i = 0; i < N; i++) {
        printf("A %d. atlag: ", i + 1);
        scanf("%lf", &jegyek[i]);
    }

    i = 0;
    while (i < N && jegyek[i] >= 1.5) {
        i++;
    }
    if (i < N) {
        printf("Volt bukas!\n");
    } else {
        printf("Nem volt bukas!\n");
    }
    return EXIT_SUCCESS;
}
```

3.47.2.

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

int main() {
    double jegy;
    int sorszam = 1;
    do {
        printf("Kerem a %d. atlagot: ", sorszam++);
        scanf("%lf", &jegy);
    } while (jegy >= 1.5);
    if (jegy == 0.0 || jegy >= 1.5) {
        printf("Nem volt bukas!\n");
    } else {
        printf("Volt bukas!\n");
    }
    return EXIT_SUCCESS;
}
```

3.48.1.

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

#define N 100

int main() {
    double magassag[N];
    int i;
    for (i = 0; i < N; i++) {
        magassag[i] = (rand() % 1000) / 10.0;
    }

    int minIndex = 0;
    int maxIndex = 0;
    for (i = 0; i < N; i++) {
        if (magassag[minIndex] > magassag[i]) {
            minIndex = i;
        }
        if (magassag[maxIndex] < magassag[i]) {
            maxIndex = i;
        }
    }
    printf("A legmagasabb pont: %g, helye: %d\n", magassag[maxIndex],
maxIndex);
    printf("A legalacsonyabb pont: %g, helye: %d\n", magassag[minIndex],
minIndex);
    return EXIT_SUCCESS;
}
```

3.48.2.-3.48.4.

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

#define N 100

double magassagLekerdez() {
    static int i = 0;
    double res;
    if (i >= 0 && i <= 3)
        res = 4;
    else if (i >= 4 && i <= 10)
        res = 3;
    else if (i >= 40 && i <= 60)
        res = 7;
    else if (i >= 70 && i <= 100)
        res = 2;
    else
        res = (rand() % 1000) / 10.0;
    i++;
    return res;
}

typedef struct Siksag {
    int kezdet;
    int veg;
    int hossz;
} Siksag;
```

```
int main() {
    int i;
    int minIndex = 0;
    int maxIndex = 0;
    int emelkedoIndex;
    Siksag maxSiksag;
    Siksag aktSiksag;
    double minMagassag;
    double maxMagassag;
    double maxKulonbseg = 0;
    double magassag;
    double elozo;
    maxSiksag.hossz = 1;
    maxSiksag.veg = 0;
    for (i = 0; i < N; i++) {
        magassag = magassagLekerdez();
        if (i == 0) {
            minMagassag = magassag;
            maxMagassag = magassag;
            elozo = magassag;
        } else {
            if (abs(elozo - magassag) > maxKulonbseg) {
                maxKulonbseg = abs(elozo - magassag);
                emelkedoIndex = i;
            }
            if (minMagassag > magassag) {
                minIndex = i;
                minMagassag = magassag;
            }
            if (maxMagassag < magassag) {
                maxIndex = i;
                maxMagassag = magassag;
            }

            if (elozo == magassag) {
                if (aktSiksag.hossz == 0) {
                    aktSiksag.hossz = 2;
                    aktSiksag.kezdet = i - 1;
                    aktSiksag.veg = i;
                } else {
                    aktSiksag.veg = i;
                    aktSiksag.hossz++;
                }
            } else {
                if (aktSiksag.veg == i - 1) {
                    if (maxSiksag.hossz < aktSiksag.hossz) {
                        maxSiksag = aktSiksag;
                    }
                    aktSiksag.hossz = 0;
                }
            }

            elozo = magassag;
        }
    }
    if (maxSiksag.hossz < aktSiksag.hossz && aktSiksag.hossz > 1) {
        maxSiksag = aktSiksag;
    }
```

```
    double maxFok = atan(maxKulonbseg / 100.0) / (M_PI / 180.0);
    printf("A legmagasabb pont: %g, helye: %d\n", maxMagassag, maxIndex);
    printf("A legalacsonyabb pont: %g, helye: %d\n", minMagassag,
minIndex);
    printf("A legmeredekebb emelkedo: %g fok, helye: %d\n", maxFok,
emelkedoIndex);

    printf("A leghosszabb siksag adatai:\n\tkezdete: %d\n\tvege:
%d\n\thossza: %d\n",
            maxSiksag.kezdet, maxSiksag.veg, maxSiksag.hossz);
    return EXIT_SUCCESS;
}
```

3.49.1-3.49.3.

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

typedef struct Elem {
    double value;
    int index;
    struct Elem * next;
} Elem;

typedef struct Vector {
    Elem * fej;
} Vector;

void initVector(Vector * vector) {
    vector->fej = NULL;
}

double lekerdezVector(Vector vector, int index) {
    Elem * tmp = vector.fej;
    while (tmp && tmp->index < index)
        tmp = tmp->next;
    if (tmp && tmp->index == index)
        return tmp->value;
    return 0.0;
}

int torolVector(Vector * vector, int pos) {
    Elem * prv = 0, * tmp = vector->fej;
    while (tmp && tmp->index < pos) {
        prv = tmp;
        tmp = tmp->next;
    }
    if (tmp && tmp->index == pos) {
        if (prv)
            prv->next = tmp->next;
        else
            vector->fej = tmp->next;
        free(tmp);
        return 1;
    }
    return 0;
}

void tisztitVector(Vector * vector) {
```

```
    Elem * tmp = vector->fej, * prv = 0;
    while (tmp) {
        if (tmp->value == 0) {
            Elem * tmp2 = tmp->next;
            torolVector(vector, tmp->index);
            tmp = tmp2;
        } else {
            prv = tmp;
            tmp = tmp->next;
        }
    }
}

void felszabaditVector(Vector * vector) {
    Elem * tmp1 = vector->fej;
    Elem * tmp2;
    while (tmp1) {
        tmp2 = tmp1->next;
        free(tmp1);
        tmp1 = tmp2;
    }
    vector->fej = 0;
}

void mutatVector(Vector vector) {
    int i, dim;
    if ((dim = legutolsoVector(vector)) == 0) {
        printf("ures\n");
        return;
    }
    for (i = 0; i <= dim; i++) {
        printf("%lg", lekerdezVector(vector, i));
        if (i < dim)
            printf(",");
    }
    printf("\n");
}

void beallitVector(Vector * vector, double value, int pos) {
    if (value == 0.0)
        return;
    if (!vector->fej) {
        vector->fej = (Elem*) malloc(sizeof (Elem));
        vector->fej->value = value;
        vector->fej->index = pos;
        vector->fej->next = 0;
    } else {
        Elem * prv = NULL, * tmp = vector->fej;
        while (tmp && (tmp->index < pos)) {
            prv = tmp;
            tmp = tmp->next;
        }
        if (tmp && tmp->index == pos) {
            tmp->value = value;
            return;
        }
        Elem * nm = (Elem*) malloc(sizeof (Elem));
        nm->value = value;
        nm->index = pos;
```

```
        if (!tmp) {
            prv->next = nm;
            prv->next->next = 0;
        } else {
            if (prv) {
                prv->next = nm;
                prv->next->next = tmp;
            } else {
                prv = nm;
                prv->next = tmp;
                vector->fej = prv;
            }
        }
    }
}

void rendezVector(Vector vector) {
    if (!vector.fej)
        return;
    Elem * tmp1, * tmp2, * min;
    double temp;
    tmp1 = vector.fej;
    while (tmp1->next) {
        min = tmp2 = tmp1;
        while (tmp2) {
            if (tmp2->value < min->value)
                min = tmp2;
            tmp2 = tmp2->next;
        }
        temp = min->value;
        min->value = tmp1->value;
        tmp1->value = temp;
        tmp1 = tmp1->next;
    }
}

double szorozVector(Vector v1, Vector v2) {
    if (!v1.fej || !v2.fej)
        return 0.0;
    double res = 0;
    Elem * tmp1 = v1.fej;
    while (tmp1) {
        Elem * tmp2 = v2.fej;
        while (tmp2 && tmp2->index < tmp1->index)
            tmp2 = tmp2->next;
        if (tmp2 && tmp2->index == tmp1->index)
            res += tmp1->value * tmp2->value;
        tmp1 = tmp1->next;
    }
    return res;
}

void addVector(Vector * v1, Vector v2) {
    Elem * tmp1 = v1->fej, * tmp2;
    while (tmp1) {
        tmp2 = v2.fej;
        while (tmp2 && tmp2->index < tmp1->index)
            tmp2 = tmp2->next;
        if (tmp2 && tmp2->index == tmp1->index)
```

```
            tmp1->value += tmp2->value;
        tmp1 = tmp1->next;
    }
    tmp1 = v2.fej;
    while (tmp1) {
        tmp2 = v1->fej;
        while (tmp2 && tmp2->index < tmp1->index)
            tmp2 = tmp2->next;
        if (!tmp2 || tmp2->index > tmp1->index)
            beallitVector(v1, tmp1->value, tmp1->index);
        tmp1 = tmp1->next;
    }
    tisztitVector(v1);
}

int meretVector(Vector vector) {
    Elem * tmp = vector.fej;
    int count = 0;
    while (tmp) {
        count++;
        tmp = tmp->next;
    }
    return count;
}

int legutolsoVector(Vector vector) {
    Elem * tmp = vector.fej;
    int dim = 0;
    while (tmp) {
        dim = tmp->index;
        tmp = tmp->next;
    }
    return dim;
}

int menuMutat() {
    int menu;
    printf("Beallit:          1\n");
    printf("Ertek:            2\n");
    printf("Mindet torol:     3\n");
    printf("Nem nullak szama: 4\n");
    printf("Legutolso:        5\n");
    printf("Megjelenit:       6\n");
    printf("Rendez:           7\n");
    printf("Hozzaad:          8\n");
    printf("Szoroz:           9\n");
    printf("Kilepes:         10\n");
    scanf("%d", &menu);
    return menu;
}

int main() {
    Vector vector;
    initVector(&vector);
    Vector v2;
    initVector(&v2);
    beallitVector(&v2, 5, 2);
    beallitVector(&v2, 1, 6);
    beallitVector(&v2, 7, 10);
```

```
    int menu;

    do {
        menu = menuMutat();
        if (menu == 1) {
            int index;
            double ertek;
            printf("Hanyadik elem? ");
            scanf("%d", &index);
            printf("Ertek? ");
            scanf("%lg", &ertek);
            beallitVector(&vector, ertek, index);
        }
        if (menu == 2) {
            int index;
            printf("Hanyadik elem? ");
            scanf("%d", &index);
            printf("A(z) %d. elem: %g\n\n", index, lekerdezVector(vector,
index));
        }
        if (menu == 3) {
            felszabaditVector(&vector);
        }
        if (menu == 4) {
            printf("Nem nullak szama: %d\n\n", meretVector(vector));
        }
        if (menu == 5) {
            printf("Legnagyobb nem nulla index: %d\n\n",
legutolsoVector(vector));
        }
        if (menu == 6) {
            mutatVector(vector);
        }
        if (menu == 7) {
            rendezVector(vector);
        }
        if (menu == 8) {
            addVector(&vector, v2);
        }
        if (menu == 9) {
            printf("A skalaris szorzat: %lg\n", szorozVector(vector,
v2));
        }
    } while (menu != 10);

    felszabaditVector(&vector);
    felszabaditVector(&v2);
    return EXIT_SUCCESS;
}
```

4.1.1.

```
#include <stdio.h>
const int listSize=7;

void printList(int head, int* listIndex, double* listData) {
    int act=head;
    printf("[");
    while (act != -1) {
```

```
        printf("%4.2lf", listData[act]);
        act = listIndex[act]; // next element
        if (act != -1)
            printf(",");
    }
    printf("]");
}

void deleteLast(int* head, int* listIndex, double* listData) {
    int act= *head, prev, prev2;
    if (act== -1)  // list already empty
        return;
    if (listIndex[act]== -1) { // list had 1 element
        *head = -1;
        return;
    }
    prev2 = act; // list had more elements
    act = listIndex[act];
    prev = act; // list had more elements
    act = listIndex[act];
    while (act != -1) {
        prev2 = prev;
        prev = act;
        act = listIndex[act]; // next element
    }
    // now act == -1 (pass the end pointer)
    // prev - pointer of the last element
    // prev2 - pointer of last but one element
    listIndex[prev2] = -1;
}

int main() {
    double listData[listSize]={34, 12.55, 893.2, 2, 11.6, 47.5, 45.3};
    int listIndex[listSize]={-1, 0, 4, 1, 6, 3, 5};
    int head=2;

    printf("list: ");
    printList(head, listIndex, listData);
    printf("\n");
    deleteLast(&head, listIndex, listData);
    printList(head, listIndex, listData);
    printf("\n");
    deleteLast(&head, listIndex, listData);
    printList(head, listIndex, listData);
    printf("\n");
    deleteLast(&head, listIndex, listData);
    printList(head, listIndex, listData);
    printf("\n");
    deleteLast(&head, listIndex, listData);
    printList(head, listIndex, listData);
    printf("\n");
    deleteLast(&head, listIndex, listData);
    printList(head, listIndex, listData);
    printf("\n");
    deleteLast(&head, listIndex, listData);
    printList(head, listIndex, listData);
    printf("\n");
    deleteLast(&head, listIndex, listData);
    printList(head, listIndex, listData);
```

```
    printf("\n");
    deleteLast(&head, listIndex, listData);
    printList(head, listIndex, listData);
    return 0;
}
// Ha törlés után elemet akarunk beszúrni,
// akkor célszerű külön listában tárolni a szabad elemeket
```

4.1.2.

```
#include <stdio.h>
#include <stdlib.h>
const int listSize = 7;

void printList(int head, int* listIndex, double* listData) {
    int act = head;
    printf("[");
    while (act != -1) {
        printf("%4.2lf", listData[act]);
        act = listIndex[act]; // next element
        if (act != -1)
            printf(", ");
    }
    printf("]\n");
}

void deleteLast(int* head, int* listIndex, double* listData) {
    int act = *head, prev, prev2;
    if (act == -1)  // list already empty
        return;
    if (listIndex[act] == -1) { // list had 1 element
        *head = -1;
        return;
    }
    prev2 = act; // list had more elements
    //act = listIndex[act];
    prev = act = listIndex[act]; // list had more elements
    act = listIndex[act];
    while (act != -1) {
        prev2 = prev;
        prev = act;
        act = listIndex[act]; // next element
    }
    // now act == -1 (pass the end pointer)
    // prev - pointer of the last element
    // prev2 - pointer of last but one element
    listIndex[prev2] = -1;
}

void deleteFirst(int* head, int* listIndex) {
    int toErase = *head;
    *head = listIndex[*head];
    listIndex[toErase] = -1;
}

int checkCircle(int* circle, int size) {
    int idxI;
    for(idxI = 1; idxI < size; ++idxI) {
        if(circle[0] == circle[idxI])
            return 1;
```

```
    }
    return 0;
}

int checkConsistent(int * listIndex, int size, int head) {
    int idxI, validSize = 0;
    int act = head;
    int* circle = (int*)malloc(sizeof (int)*size);
    for(idxI = 0; idxI < size; ++idxI) {
       if(listIndex[idxI] != -1) {
            validSize++;
       }
    }
    if(head == -1 || validSize == 0) {
        printf("List is inconsistent.\n");
        return 1;
    }
    for (idxI = 0; idxI < validSize; ++idxI) {
       if(listIndex[act] == -1) {
          printf("List is inconsistent.\n");
          return 1;
       }
        act = listIndex[act];
        circle[idxI] = act;
    }
    for(idxI = 0; idxI < validSize; ++idxI)
        if(checkCircle(&circle[idxI], validSize - idxI)) {
            printf("List is inconsistent.\n");
            return 1;
        }
    printf("List is consistent.\n");
    return 0;
}
int main() {
    double listData[7] = {34, 12.55, 893.2, 2, 11.6, 47.5, 45.3};
    int listIndex[7] = {-1, 0, 4, 1, 6, 3, 5};
    int head = 2;

    deleteFirst(&head, listIndex);
    if(!checkConsistent(listIndex,
sizeof(listIndex)/sizeof(listIndex[0]), head)) {
       printf("list: ");
       printList(head, listIndex, listData);
    }
    return 0;
}
```

4.1.3.

```
#include <stdio.h>
#include <stdlib.h>
const int listSize = 7;

void printList(int head, int* listIndex, double* listData, int size) {
    int act = head, idxI;
    printf("[");
    for(idxI = 0; idxI < size; ++idxI){
       printf("%4.2lf", listData[act]);
       act = listIndex[act]; // next element
        if (idxI != size -1)
```

```
            printf(", ");
    }
    printf("]\n");
}
int main() {
    double listData[7] = {34, 12.55, 893.2, 2, 11.6, 47.5, 45.3};
    int listIndex[7] = {1, 2, 3, 4, 5, 6, 0};
    int head;

    printf("From where should the cyclic list start? (0-6) ");
    scanf("%d", &head);
     printf("list: ");
     printList(head, listIndex, listData,
sizeof(listData)/sizeof(listData[0]));
    return 0;
}
```

4.2.1.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
const int listSize=5;
typedef struct {
    char name[100];
    int age;
    int next;
} Element;

void printList(int head, Element* myList) {
    int act=head;
    printf("[");
    while (act != -1) {
        printf("(%s, %d)", myList[act].name, myList[act].age);
        act = myList[act].next; // next element
        if (act != -1)
            printf(", ");
    }
    printf("]");
}

int freeSpace(int listSize, Element* myList) {
    int trial;
    srand(333);
    do { // if list is full then this is infinite loop
        trial = rand()%(listSize-1)+1;
    } while (myList[trial].next != -2);
    return trial;
}

void addFirst(int* head, int listSize, Element* myList) {
    int oldHead= *head, newHead;
    char name[100];
    int age;

    printf("name: ");
    scanf("%s", name);
    printf("age: ");
    scanf("%d", &age);
    newHead = freeSpace(listSize, myList);
```

```
    myList[newHead].next = oldHead;
    *head = newHead;
    strcpy(myList[newHead].name, name);
    myList[newHead].age = age;
}

int main() {
    Element myList[listSize]={{ "", 0, -2}, {"", 0, -2}, {"", 0, -2},
{"", 0, -2}, {"", 0, -2}};
    int head=-1; // no list

    printf("list: ");
    printList(head, myList);
    printf("\n");
    addFirst(&head, listSize, myList);
    printList(head, myList);
    printf("\n");
    addFirst(&head, listSize, myList);
    printList(head, myList);
    printf("\n");
    addFirst(&head, listSize, myList);
    printList(head, myList);
    printf("\n");
    return 0;
}
```

4.2.2.–4.2.3.

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

typedef struct {
    char name[100];
    int age;
    int next;
} Element;

void printList(int head, Element* myList) {
    int act=head;
    printf("[");
    while (act != -1) {
        printf("(%s, %d)", myList[act].name, myList[act].age);
        act = myList[act].next; // next element
        if (act != -1)
            printf(", ");
    }
    printf("]");
}

int checkFreeSpace(int * indexArray, int listSize){
    int idxI;

    for(idxI = 0; idxI < listSize; ++idxI){
        if(indexArray[idxI] == -2) //van még hely
            return 0;
    }
    return 1;
}
```

```
void reAllocation(Element* myList, int* listSize) {
    int idxI;
    Element* myList2;
    printf("Array is full, reallocating.\n");
    myList2 = (Element*)malloc(sizeof(Element)*(2*(*listSize)));
    for(idxI = 0; idxI < *listSize; ++idxI) {
        myList2[idxI] = myList[idxI];
    }
    for(idxI = *listSize; idxI < (2 * (*listSize)); ++idxI) {
        myList2[idxI].next = -2;
    }
    myList = myList2;
    *listSize = 2 * (*listSize);

}

int freeSpace(int listSize, Element* myList) {
    int trial, idxI;
    int * indexArray;
    indexArray = (int*)malloc(sizeof(int) * listSize);
    for(idxI = 0; idxI < listSize; ++idxI) {
        indexArray[idxI] = -2;
    }
    srand(time(NULL));
    do { // egészen addig fut, amíg üres helyet nem talál, ha nincs ilyen
meghívja a reAllocation fv-t
        if(checkFreeSpace(indexArray, listSize)) {
            reAllocation(myList, &listSize);
            indexArray = (int*)malloc(sizeof(int) * listSize);
            for(idxI = 0; idxI < listSize; ++idxI) {
                indexArray[idxI] = -2;
            }
        }
        trial = rand()%(listSize);
        indexArray[trial] = myList[trial].next;
    } while (myList[trial].next != -2);
    free(indexArray);
    return trial;
}

void addLast(int* sentinel, int listSize, Element* myList) {
    int newSentinel;
    char name[100];
    int age;

    printf("name: ");
    scanf("%s", name);
    printf("age: ");
    scanf("%d", &age);
    newSentinel = freeSpace(listSize, myList);
    myList[*sentinel].next = newSentinel;
    myList[newSentinel].next = -1;
    strcpy(myList[newSentinel].name, name);
    myList[newSentinel].age = age;
    *sentinel = newSentinel;
}

void addFirst(int* sentinel, int listSize, Element* myList) {
```

```
    int newSentinel;
    char name[100];
    int age;

    printf("name: ");
    scanf("%s", name);
    printf("age: ");
    scanf("%d", &age);
    newSentinel = freeSpace(listSize, myList);
    myList[newSentinel].next = -1;
    strcpy(myList[newSentinel].name, name);
    myList[newSentinel].age = age;
    *sentinel = newSentinel;
}

int main() {
    int idxI, listSize = 3;
    int head=-1; // no list
    int sentinel;

    Element* myList = (Element*)malloc(sizeof(Element)*listSize);
    for(idxI = 0; idxI < listSize; ++idxI) {
        myList[idxI].next = -2;
    }
    printf("list: ");
    printList(head, myList);
    printf("\n");
    addFirst(&head, listSize, myList);
    printList(head, myList);
    printf("\n");
    sentinel = head;
    for(idxI = 0; idxI < 11; ++idxI) {
        addLast(&sentinel, listSize, myList);
        printList(head, myList);
        //printf("%d", myList[idxI].next);
        printf("\n");
    }
    free(myList);
    return 0;
}
```

4.3.1.

```
#include <stdio.h>
#include <malloc.h>
typedef struct le {
    int data;
    struct le* next;
} listElem;
void displayElement(listElem* act) {
    printf("[%d, %#p]->", act->data, act->next);
}

int main() {
    listElem *act=NULL, *head=NULL, elem1={12, NULL}, elem2={45, NULL},
elem3={7, NULL};

    head = &elem1;
    elem1.next = &elem2;
    elem2.next = &elem3;
```

```
    elem3.next = NULL; // list finished
    act = head;
    printf("elements:");
    displayElement(act);
    act = act->next;
    displayElement(act);
    act = act->next;
    displayElement(act);
    printf("NULL");
    printf("\n%9s%-#18p%-#18p%-#15p%-18s", "address:", head, head->next,
head->next->next, "");
    printf("\n\n%9s%-#18p", "head:", head);
    printf("\n%9s%-#18p\n\n", "address:", &head);
    return 0;
}
```

4.3.2.

```
#include <stdio.h>
#include <malloc.h>
typedef struct le {
    int data;
    struct le* next;
    struct le* prev;
} listElem;
void displayElement(listElem* act) {
    printf("[%d, %#p]->", act->data, act->next);
}
void displayElement2(listElem* act) {
    printf("[%d, %#p]->", act->data, act->prev);
}

int main() {
    listElem *act=NULL, *head=NULL, *tail=NULL, elem1={12, NULL, NULL},
elem2={45, NULL, NULL}, elem3={7, NULL, NULL};

    head = &elem1;
    //head->prev = NULL;
    elem1.next = &elem2;
    elem1.prev = NULL;
    elem2.next = &elem3;
    elem2.prev = &elem1;
    elem3.next = NULL; // list finished
    elem3.prev = &elem2;
    act = head;
    printf("elements:");
    displayElement(act);
    act = act->next;
    displayElement(act);
    act = act->next;
    tail = act;
    displayElement(act);
    printf("NULL");
    printf("\n%9s%-#18p%-#18p%-#15p%-18s", "address:", head, head->next,
head->next->next, "");
    printf("\n\n%9s%-#18p", "head:", head);
    printf("\n%9s%-#18p\n\n", "address:", &head);

    printf("Now let's see backward: \n");
    displayElement2(tail);
```

```
    displayElement2(tail->prev);
    displayElement2(tail->prev->prev);
    printf("NULL\n");
    return 0;
}
```

4.4.1.

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

typedef struct le {
    char name[100];
    int price;
    float power;
    struct le* next;
} listElem;

listElem* createListElem() {
    listElem* result=(listElem*)malloc(sizeof(listElem));
    printf("name: ");
    scanf("%s", result->name);
    printf("price: ");
    scanf("%d", &result->price);
    printf("power: ");
    scanf("%f", &result->power);
    result->next = NULL;
    return result;
}

void displayListElem(listElem* myList) {
    printf("(name: %s, ", myList->name);
    printf("price: %d, ", myList->price);
    printf("power: %f)", myList->power);
}

void deleteList(listElem* sentinel) {
    listElem *act, *prev;
    act = sentinel->next;
    while (act) {
        prev = act;
        act = act->next;
        free(prev);
    }
}

void listList(listElem* sentinel) {
    listElem *act=sentinel->next;
    printf("List elments: ");
    while (act) {
        displayListElem(act);
        act = act->next;
        if (act)
            printf(",");
    }
    printf("\n");
}

void insertAfter(listElem* sentinel, listElem* myElem, int place) {
```

```
    // place=0 means insert as the first element
    listElem *act=sentinel->next, *prev=sentinel;
    while (place>0 && act!=NULL) {
        prev = act;
        act = act->next;
        place--;
    }
    prev->next = myElem;
    myElem->next = act;
}

int main() {
    listElem sentinel={"", 0, 0, NULL}, *temp=NULL;
    temp = createListElem();
    insertAfter(&sentinel, temp, 2);
    listList(&sentinel);

    temp = createListElem();
    insertAfter(&sentinel, temp, 2);
    listList(&sentinel);

    temp = createListElem();
    insertAfter(&sentinel, temp, 2);
    listList(&sentinel);

    temp = createListElem();
    insertAfter(&sentinel, temp, 2);
    listList(&sentinel);
    deleteList(&sentinel);
    return 0;
}
```

4.4.2.

```
void deleteAt(listElem* sentinel, int place) {
    // place=0 means insert as the first element
    listElem *act=sentinel->next, *prev=sentinel;
    place--; // listElement numbering start from 1 (sentinel is 0)
    while (place>0 && act!=NULL) {
        prev = act;
        act = act->next;
        place--;
    }
    if (act == NULL) return; // the same condition as: place =! 0
    prev->next = act->next;
    free(act); act=NULL;
}
```

4.4.3.

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

typedef struct le {
    char name[100];
    int price;
    float power;
    struct le* next;
} listElem;
listElem* createListElem() {
```

```
    listElem* result=(listElem*)malloc(sizeof(listElem));
    printf("name: ");
    scanf("%s", result->name);
    printf("price: ");
    scanf("%d", &result->price);
    printf("power: ");
    scanf("%f", &result->power);
    result->next = NULL;
    return result;
}

void displayListElem(listElem* myList) {
    printf("(name: %s, ", myList->name);
    printf("price: %d, ", myList->price);
    printf("power: %f)", myList->power);
}

void deleteList(listElem* sentinel) {
    listElem *act, *prev;
    act = sentinel->next;
    while (act) {
        prev = act;
        act = act->next;
        free(prev);
    }
}

void listList(listElem* sentinel) {
    listElem *act=sentinel->next;
    printf("List elments: ");
    while (act) {
        displayListElem(act);
        act = act->next;
        if (act)
            printf(", ");
    }
    printf("\n");
}

void insertAfter(listElem* sentinel, listElem* myElem, int place, int
*size) {
    // place=0 means insert as the first element
    listElem *act = sentinel->next, *prev = sentinel;
    while (place > 0 && act != NULL) {
        prev = act;
        act = act->next;
        place--;
    }
    prev->next = myElem;
    myElem->next = act;
    (*size)++;
}

void invertElements(listElem* sentinel, int invert_1, int invert_2) {
    listElem *act, *inv1, *inv2, *temp, *prev1, *prev2, *next1, *next2,
*tempPrev, *tempNext;
    int idxI;
    act = inv1 = inv2 = prev1 = prev2 = next1 = next2 = tempPrev =
tempNext = NULL;
```

```
    act = sentinel->next;
    tempPrev = sentinel;
    tempNext = act->next;
    for(idxI = 1; idxI <= invert_1; ++idxI) {
        if (idxI == invert_1) {
             inv1 = act;
            prev1 = tempPrev;
            next1 = tempNext;
        }
        else {
            tempPrev = act;
            act = act->next;
            tempNext = act->next;
        }
    }
    act = sentinel->next;
    tempPrev = sentinel;
    tempNext = act->next;
    for(idxI = 1; idxI <= invert_2; ++idxI) {
        if (idxI == invert_2) {
             inv2 = act;
            prev2 = tempPrev;
            next2 = tempNext;
        }
        else {
            tempPrev = act;
            act = act->next;
            tempNext = act->next;
        }
    }
    if((invert_1 - invert_2) * (invert_1 - invert_2) == 1) {
        if(invert_1 < invert_2) {
            prev1->next = inv2;
            inv1->next = inv2->next;
            inv2->next = inv1;
        }
        else if(invert_2 < invert_1) {
            prev2->next = inv1;
            inv2->next = inv1->next;
            inv1->next = inv2;
        }
    }
    else if(inv1 != NULL && inv2 != NULL) {
        prev1->next = inv2;
        inv2->next = next1;
        prev2->next = inv1;
        inv1->next = next2;


    }
    else
        printf("Cannot invert!\n");

}

void listFromEnd(listElem* sentinel, int size) {
    listElem* act = sentinel->next;
    int idxI;
```

```
    for(idxI = 0; idxI < size; ++idxI) {
        if(idxI == size-1) {
            printf(" (name: %s, ", act->name);
            printf("price: %d, ", act->price);
            printf("power: %f)", act->power);
            if(act == sentinel->next)
                printf("\n");
            else
                printf(",");
            listFromEnd(sentinel, size-1);

        }
        else
            act = act->next;
    }
}

void searchbyName(listElem* sentinel, int size, int searchAfter, int
searchAll, char* name) {
    listElem *act = sentinel->next;
    int idxI;

    printf("Found:\n");
    for(idxI = 0; idxI < searchAfter; ++idxI)
        act = act->next;
    while(act != NULL) {
        if(!strcmp(act->name, name))
            printf("(name: %s, price: %d, power: %f)\n", act->name, act-
>price, act->power);
        if(!searchAll)
            return;
        else
            act = act->next;
    }
}

void searchbyPrice(listElem* sentinel, int size, int searchAfter, int
searchAll, int price) {
    listElem *act = sentinel->next;
    int idxI;

    printf("Found:\n");
    for(idxI = 0; idxI < searchAfter; ++idxI)
        act = act->next;
    while(act != NULL) {
        if(act->price == price)
            printf("(name: %s, price: %d, power: %f)\n", act->name, act-
>price, act->power);
        if(!searchAll)
            return;
        else
            act = act->next;
    }
}

void searchbyPower(listElem* sentinel, int size, int searchAfter, int
searchAll, float power) {
    listElem *act = sentinel->next;
    int idxI;
```

```
    printf("Found:\n");
    for(idxI = 0; idxI < searchAfter; ++idxI)
        act = act->next;
    while(act != NULL) {
        if(act->power == power)
            printf("(name: %s, price: %d, power: %f)\n", act->name, act-
>price, act->power);
        if(!searchAll)
            return;
        else
            act = act->next;
    }
}

void deletebyName(listElem* sentinel, int* size, char* name) {
    listElem *act = sentinel->next, *prev = sentinel;

    while(act != NULL) {
        if(!strcmp(act->name, name)) {
            prev->next = act->next;
            free(act);
            act = prev->next;
            (*size)--;
        }
        else {
            prev = act;
            act = act->next;
        }
    }
}

void deletebyPrice(listElem* sentinel, int* size, int price) {
    listElem *act = sentinel->next, *prev = sentinel;

    while(act != NULL) {
        if(act->price == price) {
            prev->next = act->next;
            free(act);
            act = prev->next;
            (*size)--;
        }
        else {
            prev = act;
            act = act->next;
        }
    }
}

void deletebyPower(listElem* sentinel, int* size, float power) {
    listElem *act = sentinel->next, *prev = sentinel;

    while(act != NULL) {
        if(act->power == power) {
            prev->next = act->next;
            free(act);
            act = prev->next;
            (*size)--;
        }
```

```
        else {
            prev = act;
            act = act->next;
        }
    }
}

int main() {
    int menu, subMenu, subSubMenu, invert_1, invert_2, listSize = 0,
searhAfter, searchPrice, deletePrice, controller, controller2,
tempSearchAfter;
    float searchPower, deletePower;
    char searchName[50], deleteName[50];
    listElem sentinel={"", 0, 0, NULL}, *temp=NULL;
    temp = createListElem();
    insertAfter(&sentinel, temp, 2, &listSize);
    listList(&sentinel);

    temp = createListElem();
    insertAfter(&sentinel, temp, 2, &listSize);
    listList(&sentinel);

    temp = createListElem();
    insertAfter(&sentinel, temp, 2, &listSize);
    listList(&sentinel);

    /*temp = createListElem();
    insertAfter(&sentinel, temp, 2, &listSize);
    listList(&sentinel);*/
    for(;;) {
        printf("\n***Menu***\n");
        printf("1. Search\n2. Invert two elements\n3. List from the
end\n4. Delete certain element(s)\n5. Exit\n");
        scanf("%d", &menu);
        switch(menu) {
            case 1:
                controller = 0;
                for(;controller == 0;) {
                    printf("\n***Search Menu***\n1. Search the first
occurrence\n2. Search the first occurrence after index xy\n3. Search all
occurrence\n4. Back\n");
                    scanf(„%d”, &subMenu);
                    switch(subMenu) {
                        case 1:
                            controller2 = 0;
                            for(;controller2 == 0;) {
                                printf("***\nSearch the first occ.
SubMenu***\n");
                                printf("1. Search by name\n2. Search by
price\n3. Search by power\n4. Back\n");
                                scanf(„%d”, &subSubMenu);
                                switch(subSubMenu) {
                                    case 1:
                                    controller = 1;
                                    controller2 = 1;
                                        printf("Search name? ");
                                        scanf("%s", searchName);
                                        searchbyName(&sentinel, listSize,
0, 0, searchName);
```

```
                                    break;
                                case 2:
                                controller = 1;
                                    controller2 = 1;
                                    printf("Search price? ");
                                    scanf("%d", &searchPrice);
                                    searchbyPrice(&sentinel,
listSize, 0, 0, searchPrice);
                                    break;
                                case 3:
                                controller = 1;
                                    controller2 = 1;
                                    printf("Search power? ");
                                    scanf("%f", &searchPower);
                                    searchbyPower(&sentinel,
listSize, 0, 0, searchPower);
                                    break;
                                case 4:
                                    controller2 = 1;
                                    break;
                                default:
                                controller2 = 0;
                            }
                        }
                        break;
                    case 2:
                     controller2 = 0;
                        for(;controller2 == 0;) {
                            printf("\n***Search the first occ after
index SubMenu***\n");
                            printf("1. Search by name\n2. Search by
price\n3. Search by power\n4. Back\n");
                            scanf("%d", &subSubMenu);
                         if(subSubMenu == 4)
                            break;
                         for(tempSearchAfter = 0; tempSearchAfter ==
0;) {
                            printf(„After index? „);
                            scanf("%d", &searhAfter);
                            if(searhAfter <= listSize)
                                tempSearchAfter = 1;
                            else
                                printf(„Bad index!!!\n”);
                         }
                            switch(subSubMenu) {
                                case 1:
                                    controller = 1;
                                controller2 = 1;
                                    printf("Search name? ");
                                    scanf("%s", searchName);
                                    searchbyName(&sentinel, listSize,
searhAfter, 0, searchName);
                                    break;
                                case 2:
                                    controller = 1;
                                controller2 = 1;
                                    printf("Search price? ");
                                    scanf("%d", &searchPrice);
```

```
                                        searchbyPrice(&sentinel,
listSize, searhAfter, 0, searchPrice);
                                        break;
                                    case 3:
                                        controller = 1;
                                    controller2 = 1;
                                        printf("Search power? ");
                                        scanf("%f", &searchPower);
                                        searchbyPower(&sentinel,
listSize, searhAfter, 0, searchPower);
                                        break;
                                    default:
                                        controller2 = 0;
                                }
                            }
                            break;
                        case 3:
                            controller2 = 0;
                            for(;controller2 == 0;) {
                                printf("\n***Search all occ.
SubMenu***\n");
                                printf("1. Search by name\n2. Search by
price\n3. Search by power\n4. Back\n");
                                scanf("%d", &subSubMenu);
                                switch(subSubMenu) {
                                    case 1:
                                        controller = 1;
                                    controller2 = 1;
                                        printf("Search name? ");
                                        scanf("%s", searchName);
                                        searchbyName(&sentinel, listSize,
0, 1, searchName);
                                        break;
                                    case 2:
                                        controller = 1;
                                    controller2 = 1;
                                        printf("Search price? ");
                                        scanf("%d", &searchPrice);
                                        searchbyPrice(&sentinel,
listSize, 0, 1, searchPrice);
                                        break;
                                    case 3:
                                        controller = 1;
                                    controller2 = 1;
                                        printf("Search power? ");
                                        scanf("%f", &searchPower);
                                        searchbyPower(&sentinel,
listSize, 0, 1, searchPower);
                                        break;
                                    case 4:
                                        controller2 = 1;
                                        break;

                                    default:
                                        controller2 = 0;
                                }
                            }
                            break;
                    }
```

```
                    if(subMenu == 4)
                        break;
                }
                break;
            case 2:
                for(;;) {
                    printf("\nWhich two would you like to invert? (e.g. 1
4)\n");
                    scanf("%d%d", &invert_1, &invert_2);
                    if(listSize < invert_1 || listSize < invert_2 ||
invert_1 == 0 || invert_2 == 0)
                        printf("Bad index(es)!!!\n");
                    else {
                        invertElements(&sentinel, invert_1, invert_2);
                        break;
                    }
                }
                break;
            case 3:
              printf("List elements: ");
                listFromEnd(&sentinel, listSize);
                //listList(&sentinel);
                break;
            case 4:
                controller = 0;
                for(;controller == 0;) {
                    printf("\n*** Delete SubMenu***\n");
                    printf("1. Delete by name\n2. Delete by price\n3.
Delete by power\n4. Back\n");
                    scanf("%d", &subMenu);
                    switch(subMenu) {
                        case 1:
                            printf("Delete name? ");
                            scanf("%s", deleteName);
                            deletebyName(&sentinel, &listSize,
deleteName);
                            controller = 1;
                            break;
                        case 2:
                            printf("Delete price? ");
                            scanf("%d", &deletePrice);
                            deletebyPrice(&sentinel, &listSize,
deletePrice);
                            controller = 1;
                            break;
                        case 3:
                            printf("Delete power? ");
                            scanf("%f", &deletePower);
                            deletebyPower(&sentinel, &listSize,
deletePower);
                            controller = 1;
                            break;
                        case 4:
                            controller = 1;
                            break;
                        default:
                            controller = 0;
                    }
                }
```

```
                break;
            case 5:
                return 0;

        }
    }
    deleteList(&sentinel);
    return 0;
}
```

4.5.1.

```
#include <iostream>
#include <cstring>
using namespace std;

class airplane {
public:
    int id;
    int passengers;
    airplane(int p1=0, int p2=0) : id(p1), passengers(p2), next(0),
prev(0) {}
    bool operator==(const airplane& para) const;
    airplane *next, *prev;
};
ostream& operator<<(ostream& os, const airplane& para) {
    os << "(id:" << para.id << ", passengers: " << para.passengers <<
")";
    return os;
}

bool airplane::operator==(const airplane& para) const {
    bool result=false;
    if (id==para.id && passengers==para.passengers)
        result=true;
    return result;
}

template <typename T1>
class list {
    T1 *head, *tail, *act;
    int elemNum;
public:
    list() : head(0), tail(0), act(0), elemNum(0) {}
    ~list();
    T1* search(const T1& para) const; // search from head to tail
    void insert(const T1& para); // before act
    void delAct(); // act moves to the previous element
    void moveAct(int para); // realative move from act
    T1& getElem() const {return *act;}
    int getElemNum() const {return elemNum;}
    template <typename U1> friend ostream& operator<<(ostream&, const
list<U1>&);
};

template <typename T1>
list<T1>::~list() {
    T1 *temp=head, *prev;
    while (temp) {
        prev=temp;
```

```
        temp=temp->next;
        delete prev;
    }
}

template <typename T1>
T1* list<T1>::search(const T1& para) const {
    // search from head to tail
    T1* temp=head;
    while (temp) {
        if (*temp==para)
            return temp;
        temp=temp->next;
    }
    return 0;
}

template <typename T1>
void list<T1>::insert(const T1& para) { // before act
    elemNum++;
    T1 *temp=new T1(para), *before;
    if (head==0) {
        head=tail=act=temp;
        return;
    }
    before=act->prev;
    if (before==0)
        head=temp;
    else
        before->next=temp;
    temp->next=act;
    act->prev=temp;
    temp->prev=before;
    act=temp; // new act value is the inserted element
}

template <typename T1>
void list<T1>::delAct() { // at act
    T1 *after, *before;
    if (act==0)
        return;
    before=act->prev;
    after=act->next;
    if (before==0)
        act=head=after;
    else {
        before->next=after;
        act=before;
    }
    if (after==0)
        tail=before;
    else
        after->prev=before;
    elemNum--;
}

template <typename T1>
void list<T1>::moveAct(int para) {
    if (para>0)
```

```
        while (act->next && para) {
            act=act->next;
            para--;
        }
    else
        while (act->prev && para) {
            act=act->prev;
            para++;
        }
}

template <typename U1>
ostream& operator<< (ostream& os, const list<U1>& para) {
    U1* temp=para.head;
    if (!temp)
        os << "empty";
    while (temp) {
        os << *temp << ", ";
        temp=temp->next;
    }
    os << endl;
    return os;
}

int main() {
    airplane a(1, 500), b(2, 450), c(3, 220), d(4, 260);
    list<airplane> myList;
    cout << myList;
    myList.insert(a);
    cout << myList;
    myList.insert(b);
    cout << myList;
    myList.insert(c);
    cout << myList;
    myList.insert(d);
    cout << myList;
    cout << "ElemNum:" << myList.getElemNum() << endl;
    cout << "act elem: " << myList.getElem() << endl;
    cout << "moveAct(2)" << endl;
    myList.moveAct(2);
    cout << "act elem: " << myList.getElem() << endl;
    cout << "delAct()" << endl;
    myList.delAct();
    cout << myList;
    cout << "moveAct(-2)" << endl;
    myList.moveAct(-2);
    cout << "act elem: " << myList.getElem() << endl;
    cout << "delAct()" << endl;
    myList.delAct();
    cout << "act elem: " << myList.getElem() << endl;
    cout << myList;
    return 0;
}
```

4.5.2.

```
#include <iostream>
#include <cstring>
using namespace std;
```

```
class airplane {
public:
    int id;
    int passengers;
    airplane(int p1=0, int p2=0) : id(p1), passengers(p2), next(0),
prev(0) {}
    bool operator==(const airplane& para) const;
    airplane *next, *prev;
};
ostream& operator<<(ostream& os, const airplane& para) {
    os << "(id:" << para.id << ", passengers: " << para.passengers <<
")";
    return os;
}

bool airplane::operator==(const airplane& para) const {
    bool result=false;
    if (id==para.id && passengers==para.passengers)
        result=true;
    return result;
}

template <typename T1>
class list {
    T1 *head, *tail, *act;
    int elemNum;
public:
    list() : head(0), tail(0), act(0), elemNum(0) {}
    ~list();
    T1* search(const T1& para) const; // search from head to tail
    void insert(const T1& para); // before act
    void delAct(); // act moves to the previous element
    void delAbs(int para);
    void moveAct(int para); // realative move from act
    void moveAbs(int para);
    T1& getElem() const {return *act;}
    int getElemNum() const {return elemNum;}
    template <typename U1> friend ostream& operator<<(ostream&, const
list<U1>&);
};

template <typename T1>
list<T1>::~list() {
    T1 *temp=head, *prev;
    while (temp) {
        prev=temp;
        temp=temp->next;
        delete prev;
    }
}

template <typename T1>
T1* list<T1>::search(const T1& para) const {
    // search from head to tail
    T1* temp=head;
    while (temp) {
        if (*temp==para)
            return temp;
        temp=temp->next;
```

```
    }
    return 0;
}

template <typename T1>
void list<T1>::insert(const T1& para) { // before act
    elemNum++;
    T1 *temp=new T1(para), *before;
    if (head==0) {
        head=tail=act=temp;
        return;
    }
    before=act->prev;
    if (before==0)
        head=temp;
    else
        before->next=temp;
    temp->next=act;
    act->prev=temp;
    temp->prev=before;
    act=temp; // new act value is the inserted element
}

template <typename T1>
void list<T1>::delAbs(int para) {
  moveAbs(para);
  delAct();
}

template <typename T1>
void list<T1>::delAct() { // at act
    T1 *after, *before;
    if (act==0)
        return;
    before=act->prev;
    after=act->next;
    if (before==0)
        act=head=after;
    else {
        before->next=after;
        act=before;
    }
    if (after==0)
        tail=before;
    else
        after->prev=before;
    elemNum--;
}

template <typename T1>
void list<T1>::moveAct(int para) {
    if (para>0)
        while (act->next && para) {
            act=act->next;
            para--;
        }
    else
        while (act->prev && para) {
            act=act->prev;
```

```
            para++;
        }
}

template <typename T1>
void list<T1>::moveAbs(int para) {
  act=head;
  for (int i=para;i>0;i--) {
    act=act->next;
  }
}

template <typename U1>
ostream& operator<< (ostream& os, const list<U1>& para) {
    U1* temp=para.head;
    if (!temp)
        os << "empty";
    while (temp) {
        os << *temp << ", ";
        temp=temp->next;
    }
    os << endl;
    return os;
}

int main() {
    airplane a(1, 500), b(2, 450), c(3, 220), d(4, 260);
    list<airplane> myList;
    cout << myList;
    myList.insert(a);
    cout << myList;
    myList.insert(b);
    cout << myList;
    myList.insert(c);
    cout << myList;
    myList.insert(d);
    cout << myList;
    cout << "ElemNum:" << myList.getElemNum() << endl;
    cout << "act elem: " << myList.getElem() << endl;
    cout << "moveAct(2)" << endl;
    myList.moveAct(2);
    cout << "act elem: " << myList.getElem() << endl;
    cout << "delAct()" << endl;
    myList.delAct();
    cout << myList;
    cout << "moveAct(-2)" << endl;
    myList.moveAct(-2);
    cout << "act elem: " << myList.getElem() << endl;
    cout << "delAct()" << endl;
    myList.delAct();
    cout << "act elem: " << myList.getElem() << endl;
    cout << myList;
    return 0;
}
```

4.6.1.

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

```
struct TElement
{
  struct TElement * next;
  int Data;
};

struct TList
{
  struct TElement * Head;
  struct TElement * Actual;
};

void InitList(struct TList * L)
{
  L->Head = L->Actual = NULL;
}

// we suppose that, the actual is the last element
void Add(struct TList * L, int data)
{
  if (L->Head == NULL)
    L->Head = L->Actual = (struct TElement *)malloc(sizeof(struct
TElement));
  else
  {
    L->Actual->next = (struct TElement *)malloc(sizeof(struct TElement));
    L->Actual = L->Actual->next;
  }
  L->Actual->Data = data;
  L->Actual->next = NULL;
}

void PrintList(struct TList * L)
{
  if (L->Head == NULL)
    return;
  L->Actual = L->Head;
  printf("The list: ");
  while (L->Actual->next)
  {
    printf("%d ", L->Actual->Data);
    L->Actual = L->Actual->next;
  }
  printf("%d\n", L->Actual->Data);
}

void FreeList(struct TList * L)
{
  struct TElement * tmp;
  if (L->Head == NULL)
    return;
  L->Actual = L->Head;
  do
  {
    tmp = L->Actual;
    L->Actual = L->Actual->next;
    free(tmp);
    tmp = NULL;
  } while (L->Actual);
```

```
}

int main()
{
  struct TList List;
  int num;
  InitList(&List);
  do
  {
    printf("Type a number: ");
    scanf("%d", &num);
    if (num != 0)
    {
      if (num != -1)
        Add(&List, num);
      else
        PrintList(&List);
    }
  } while (num != 0);
  FreeList(&List);
  return 0;
}
```

4.7.1.

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

struct PrimeArray {
    int current_number;
    struct PrimeArray* next;
};

void UploadPrimeArray(struct PrimeArray*, int);
void ListPrimeArray(struct PrimeArray);
void RemoveNotPrimes(struct PrimeArray*);
void FreePrimeArray(struct PrimeArray*);

int main() {
    int MAX_NUMBER;
    struct PrimeArray my_prime_array;

    printf("Pleas enter a number, the orignal list will be generated from
2 to this number.\n");
    scanf("%d", &MAX_NUMBER);
    printf("The original list:\n");
    UploadPrimeArray(&my_prime_array, MAX_NUMBER);
    ListPrimeArray(my_prime_array);
    RemoveNotPrimes(&my_prime_array);
    printf("\n\nThe list, after removing not primes:\n");
    ListPrimeArray(my_prime_array);
    FreePrimeArray(&my_prime_array);
    return 0;
}

void UploadPrimeArray(struct PrimeArray* first, int max_number) {
    int i;
    struct PrimeArray* actual;

    actual = first;
```

```
    actual->current_number = 2;
    actual->next = NULL;
    //létrehozok egy láncolt listát, aminek elemei 2-től n-ig a számok
    for (i = 3; i <= max_number; ++i) {
        actual->next = (struct PrimeArray*)malloc(sizeof(struct
PrimeArray));
        actual = actual->next;
        actual->current_number = i;
        actual->next = NULL;
    }

}
void ListPrimeArray(struct PrimeArray first) {
    int i = 1;
    struct PrimeArray* actual = &first;

    while (actual) {
        printf("%3d.element: %3d\n", i++, actual->current_number);
        actual = actual->next;
    }

}
void RemoveNotPrimes(struct PrimeArray* first) {
    struct PrimeArray* actual, *prev, *next;

    //dupla ciklus, a külsővel megyek végig a listán
    while (first->next) {
        actual =  first->next;
        prev = first;
        next = actual->next;

        //a belső ciklussal vizsgálom az akutális első elem utáni számok
oszthatóságát
        while (actual->next) {
            if ((actual->current_number % first->current_number) == 0) {
                free(actual);
                actual = next;
                next = actual->next;
                prev->next = actual;
            }
            else {
                prev = actual;
                actual = actual->next;
                next = actual->next;
            }
        }
        printf("\nInner steps (%%%d)\n", first->current_number);
        ListPrimeArray(*first);
        first = first->next;
    }
}
void FreePrimeArray(struct PrimeArray* first) {
    //a lista első eleme statikus, ezért a másodiktól kezdve szabadítunk
fel
    struct PrimeArray* actual = first->next;

    while (actual) {
        first = actual->next;
        free(actual);
```

```
        actual = first;
    }
}
```

4.8.1.

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

struct array {
    int current_number;
    struct array* next;
};

void Uploadarray(struct array*, int*, int);
void Combarrays(struct array**, struct array**);
void Listarray(struct array*);
void FreeArray(struct array*);

int main() {
    struct array * head_first,* head_second;
    int set_1[] = {1, 2, 4, 5, 6, 8, 9, 10, 13};
    int set_2[] = {0, 2, 3, 5, 6, 7, 9, 11, 12, 13};
    head_first = (struct array*)malloc(sizeof(struct array));
    head_second = (struct array*)malloc(sizeof(struct array));

    Uploadarray(head_first, set_1, sizeof(set_1)/sizeof(int));
    Uploadarray(head_second, set_2, sizeof(set_2)/sizeof(int));
    printf("First list:\n");
    Listarray(head_first);
    printf("\nSecond list:\n");
    Listarray(head_second);
    Combarrays(&head_first, &head_second);
    printf("\nFirst list after creating the union of the sets:\n");
    Listarray(head_first);
    FreeArray(head_first);
    return 0;
}

void Uploadarray(struct array* first, int* array_here, int size) {
    int i;
    struct array* actual;

    actual = first;
    actual->current_number = array_here[0];
    actual->next = NULL;
    for(i = 1; i < size; i++) {
        actual->next = (struct array*)malloc(sizeof(struct array));
        actual = actual->next;
        actual->current_number = array_here[i];
        actual->next = NULL;
    }
}
void Combarrays(struct array** first_start, struct array** second_start)
{
    struct array *first, *first_temp, *second_temp, *prev;

    //megkeresi a legkisebb elemet a két listában, ha ez a kettesben van,
akkor átálítja az első lista elejét erre, persze a lista többi eleme
marad a régi
```

```
    if ((*first_start)->current_number > (*second_start)->current_number)
{
        first_temp = *first_start;
        second_temp = *second_start;
        *second_start = (*second_start)->next;
        (*first_start) = second_temp;
        (*first_start)->next = first_temp;
    }
    prev = first = *first_start;

    //ameddig létezik a két lista, tudok rajtuk összehasonlítást végezni
    while (first && (*second_start)) {
        //ha két elem megegyezik
        if(first->current_number == (*second_start)->current_number) {
            prev = first;
            first = first->next;
            second_temp = (*second_start);
            (*second_start) = (*second_start)->next;
            free(second_temp);
        }
        //ha az első lista eleme nagyobb
        else if(first->current_number < (*second_start)->current_number) {
            prev = first;
            first = first->next;
        }
        //ha a második listában nagyobb a következő elem
        else if(first->current_number > (*second_start)->current_number) {
            second_temp = (*second_start);
            (*second_start) = (*second_start)->next;
            prev->next = second_temp;
            prev->next->next = first;
            prev = second_temp;
        }
    }
    //ha az első lista már elfogyott volna, de maradt még a másodikban
kivehető elem
    if((*second_start))
        first->next = (*second_start);
}
void Listarray(struct array* first) {
    int i = 1;
    struct array* actual = first;

    while(actual) {
        printf("%d.element: %d\n", i++, actual->current_number);
        actual = actual->next;
    }
}
void FreeArray(struct array* actual) {
    struct array* temp = actual;
    while(temp) {
        actual = temp->next;
        free(temp);
        temp = actual;
    }
}
```

4.8.2.

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

```
#include <malloc.h>

struct array {
    int current_number;
    struct array* next;
};

void UploadArray(struct array*, int*, int);
void ListArray(struct array*);
void FreeArray(struct array*);
void Intersection(struct array**, struct array*);

int main() {
    struct array* first, *second;
    struct array** ptr_first;
    int numbers_1[8] = {0,1,5,7,9,13,19,96};
    int numbers_2[8] = {1,4,9,15,19,25,37,45};

    first = (struct array*)malloc(sizeof(struct array));
    second = (struct array*)malloc(sizeof(struct array));
    ptr_first = &first;

    UploadArray(first, numbers_1, 8);
    UploadArray(second, numbers_2, 8);
    printf("The first list:\n");
    ListArray(first);
    printf("\nThe second list:\n");
    ListArray(second);
    Intersection(ptr_first, second);
    printf("\nThe first list after creating the intersecrion of the
lists:\n");
    ListArray(first);
    FreeArray(first);
    FreeArray(second);
    return 0;
}

void UploadArray(struct array* first, int* numbers, int size) {
    int i;
    struct array* actual = first;

    for(i = 0; i < size; i++) {
        if(i) {
            actual->next = (struct array*)malloc(sizeof(struct array));
            actual = actual->next;
        }
        actual->current_number = numbers[i];
        actual->next = NULL;
    }
}
void ListArray(struct array* first) {
    int i = 1;

    while(first) {
        printf("%d.element: %d\n", i++, first->current_number);
        first = first->next;
    }
}
void FreeArray(struct array* actual) {
```

```
    struct array* temp = actual;
    while(temp) {
        actual = temp->next;
        free(temp);
        temp = actual;
    }
}
void Intersection(struct array** first, struct array* second) {
    struct array* prev_first, *prev_second, *element_next;
    //az első közös elem megtalálásáig fog futni ez a ciklus, ekkor
meglesz, hogy az első listának mi is az első eleme
    while((*first) && second) {
        if((*first)->current_number > second->current_number) {
            prev_second = second;
            second = second->next;
            free(prev_second);
        }
        else if((*first)->current_number < second->current_number) {
            prev_first = *first;
            *first = (*first)->next;
            free(prev_first);
        }
        else {
            element_next = *first;
            prev_first = element_next;
            element_next = element_next->next;
            second = second->next;
            break;
        }
    }
    //további közös elemek keresése
    while(element_next && second) {
        if(element_next->current_number > second->current_number) {
            second = second->next;
        }
        else if(element_next->current_number < second->current_number) {
            prev_first->next = element_next->next;
            free(element_next);
            element_next = prev_first->next;
        }
        else {
            prev_first = element_next;
            element_next = element_next->next;
            second = second->next;
        }
    }
    //ha még maradt volna elem az első lista végén, töröljük
    if(element_next) {
        prev_first->next = NULL;
        FreeArray(element_next);
    }
}
```

5.1.1.

```
#include <iostream>
using namespace std;
const int DEFAULT_STACK_SIZE=5;
class stack {
private:
```

```
    int elemnum, store[DEFAULT_STACK_SIZE];
    bool isFull();
public:
    void Init() {elemnum=0;}
    bool isEmpty();
    bool Push(const int n);
    bool Pop(int &n);
};

bool stack::isFull() {
    return elemnum==DEFAULT_STACK_SIZE;
}
bool stack::isEmpty() {
    return elemnum==0;
}
bool stack::Push(const int n) {
    if (isFull()) return false;
    store[elemnum++]=n;
    return true;
}
bool stack::Pop(int &n) {
    if (isEmpty()) return false;
    n=store[--elemnum];
    return true;
}

int main() {
    stack *s=new stack;
    s->Init();
    char c;    int n;
    do {
        cout<<"(p)ush\np(o)p\n(e)xit\n";
        cout<<"Enter your selection : ";
        cin>>c;
        if (c=='p') {
            cout<<"Enter a number: ";
            cin>>n;
            if (s->Push(n)) cout<<"OK!\n";
        }
        if (c=='o') {
            if (s->Pop(n)) cout<<n<<'\n';
        }
    } while (c!='e');
    delete s;
    return 0;
}
```

5.1.2.

```
#include <iostream>
using namespace std;
const int DEFAULT_SIZE=1;
class stack {
private:
    int elemnum, *store, size;
    bool isFull();
public:
    void Init() {elemnum=0; store=new int[DEFAULT_SIZE];
size=DEFAULT_SIZE; }
    void increaseSize() {
```

```
      size*=2;
      cout << "increasing size -> " << size << endl;
      int* temp=new int[size];
      for (int i=0;i<elemnum;i++) {
        temp[i]=store[i];
      }
      delete[] store;
      store=temp;
    }
    bool isEmpty();
    bool Push(const int n);
    bool Pop(int &n);
};

bool stack::isFull() {
    return elemnum==size;
}
bool stack::isEmpty() {
    return elemnum==0;
}
bool stack::Push(const int n) {
    store[elemnum++]=n;
    if (elemnum==size) { increaseSize(); }
    return true;
}
bool stack::Pop(int &n) {
    if (isEmpty()) return false;
    n=store[--elemnum];
    return true;
}

int main() {
    stack *s=new stack;
    s->Init();
    char c;    int n;
    do {
        cout<<"(p)ush\np(o)p\n(e)xit\n";
        cout<<"Enter your selection : ";
        cin>>c;
        if (c=='p') {
            cout<<"Enter a number:";
            cin>>n;
            if (s->Push(n)) cout<<"OK!\n";
        }
        if (c=='o') {
            if (s->Pop(n)) cout<<n<<'\n';
        }
    } while (c!='e');
    delete s;
    return 0;
}
```

5.2.1.

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

struct TElement
{
  struct TElement * prev;
```

```
  int Value;
};

struct TStack
{
  struct TElement * Head;
  int Count;
};

void InitStack(struct TStack * S)
{
  S->Head = NULL;
  S->Count = 0;
}

void Push(struct TStack * S, int A)
{
  struct TElement * Element = (struct TElement *)malloc(sizeof(struct
TElement));
  Element->Value = A;
  if (S->Head == NULL)
  {
    Element->prev = NULL;
    S->Head = Element;
  } else
  {
    Element->prev = S->Head;
    S->Head = Element;
  }
  S->Count++;
}

int Pop(struct TStack * S)
{
  int res = 0;
  if (S->Head != NULL)
  {
    struct TElement * E = S->Head;
    res = E->Value;
    S->Head = E->prev;
    free(E);
    E = NULL;
    S->Count--;
  }
  return res;
}

void FreeStack(struct TStack * S)
{
  while (S->Head != NULL)
    Pop(S);
}

int main(int argc, char * argv[])
{
  int num;
  struct TStack Stack;
  InitStack(&Stack);
  do
```

```
  {
    printf("Type a number: ");
    scanf("%d", &num);
    if (num != 0)
      Push(&Stack, num);
  } while (num != 0);
  while (Stack.Count)
    printf("%d\n", Pop(&Stack));
  FreeStack(&Stack);
  return 0;
}
```

5.3.1.

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

struct TElement
{
  struct TElement * next;
  int Value;
};

struct TQueue
{
  struct TElement * Head;
  struct TElement * Last;
  int Count;
};

void InitQueue(struct TQueue * Q)
{
  Q->Head = NULL;
  Q->Last = NULL;
  Q->Count = 0;
}

void Push(struct TQueue * Q, int A)
{
  struct TElement * Element = (struct TElement *)malloc(sizeof(struct
TElement));
  Element->Value = A;
  Element->next = NULL;
  if (Q->Head == NULL)
  {
    Q->Head = Element;
    Q->Last = Element;
  } else
  {
    Q->Last->next = Element;
    Q->Last = Element;
  }
  Q->Count++;
}

int Pop(struct TQueue * Q)
{
  int res = 0;
  if (Q->Head != NULL)
  {
```

```
    struct TElement * E = Q->Head;
    res = E->Value;
    Q->Head = E->next;
    free(E);
    E = NULL;
    Q->Count--;
  }
  return res;
}

void FreeQueue(struct TQueue * Q)
{
  while (Q->Head != NULL)
    Pop(Q);
}

int main(int argc, char * argv[])
{
  int num;
  struct TQueue Queue;
  InitQueue(&Queue);
  do
  {
    printf("Type a number: ");
    scanf("%d", &num);
    if (num != 0)
      Push(&Queue, num);
  } while (num != 0);
  while (Queue.Count)
    printf("%d\n", Pop(&Queue));
  FreeQueue(&Queue);
  return 0;
}
```

5.4.1.

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

typedef struct Verem {
    struct Verem * kovetkezo;
    int adat;
} Verem;

void verembe(Verem ** verem, int adat) {
    Verem * uj = (Verem*)malloc(sizeof(Verem));
    uj->adat = adat;
    uj->kovetkezo = *verem;
 *verem = uj;
}

int verembol(Verem ** verem) {
    if (*verem == NULL) {
        return 0;
    }
    int adat = (*verem)->adat;
    Verem * kovetkezo = (*verem)->kovetkezo;
    free(*verem);
 *verem = kovetkezo;
    return adat;
```

```
}

int main() {
    const char * kifejezes[] = {"2", "5", "+", "3", "*", "9", "2", "1",
"+", "/", "-"};
    Verem * verem = NULL;
    int index;
    int meret = sizeof(kifejezes) / sizeof(const char*);
    for (index = 0; index < meret; index++) {
        if (kifejezes[index][0] == '+') {
            int a = verembol(&verem);
            int b = verembol(&verem);
            verembe(&verem, a + b);
        } else if (kifejezes[index][0] == '-') {
            int a = verembol(&verem);
            int b = verembol(&verem);
            verembe(&verem, b - a);
        } else if (kifejezes[index][0] == '*') {
            int a = verembol(&verem);
            int b = verembol(&verem);
            verembe(&verem, a * b);
        } else if (kifejezes[index][0] == '/') {
            int a = verembol(&verem);
            int b = verembol(&verem);
            verembe(&verem, b / a);
        } else {
            verembe(&verem, atoi(kifejezes[index]));
        }
    }
    int eredmeny = verembol(&verem);
    printf("Az eredmeny: %d\n", eredmeny);
    return EXIT_SUCCESS;
}
```

5.5.1.-5.5.2.

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

typedef struct Sor {
    struct Sor * kovetkezo;
    int adat;
} Sor;

void sorba(Sor ** sor, int adat) {
    Sor * uj = (Sor*) malloc(sizeof (Sor));
    uj->adat = adat;
    uj->kovetkezo = NULL;
    if (*sor == NULL) {
 *sor = uj;
        return;
    }
    Sor * temp = *sor;
    while (temp->kovetkezo != NULL) {
        temp = temp->kovetkezo;
    }
    temp->kovetkezo = uj;
}

int sorbol(Sor ** sor) {
```

```
    if (*sor == NULL) {
        return 0;
    }
    int adat = (*sor)->adat;
    Sor * kovetkezo = (*sor)->kovetkezo;
    free(*sor);
 *sor = kovetkezo;
    return adat;
}

void ujAzonosito(Sor ** sor) {
    int azonosito;
    Sor * temp;
    do {
        azonosito = rand() % 1000;
        temp = *sor;
        while (temp != NULL && temp->adat != azonosito) {
            temp = temp->kovetkezo;
        }
    } while (temp != NULL);
    printf("Uj azonosito: %d\n", azonosito);
    sorba(sor, azonosito);
}

void nKivesz(Sor ** sor, int n) {
    int index;
    for (index = 0; index < n && *sor != NULL; index++) {
        printf("Az %d. azonosito: %d\n", index + 1, sorbol(sor));
    }
}

int main() {
    Sor * sor = NULL;
    int menu;
    do {
        printf("Beszuras:        1\n");
        printf("Kivesz:          2\n");
        printf("Tobbet kivesz:   3\n");
        printf("Kilepes:         4\n");
        scanf("%d", &menu);
        if (menu == 1) {
            ujAzonosito(&sor);
        }
        if (menu == 2) {
            if (sor != NULL)
                printf("A kovetkezo azonosito: %d\n", sorbol(&sor));
            else
                printf("Nincs tobb varakozo!\n");
        }
        if (menu == 3) {
            int mennyi;
            printf("Mennyi azonositot szedjunk ki? ");
            scanf("%d", &mennyi);
            nKivesz(&sor, mennyi);
        }
    } while (menu != 4);
    while (sor != NULL) {
        sorbol(&sor);
    }
```

```
    return EXIT_SUCCESS;
}
```

5.6.1.-5.6.2.

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

typedef enum Rang {
    PROFESSZOR, DOCENS, ADJUNKTUS, TANARSEGED
} Rang;

typedef struct Sor {
    struct Sor * kovetkezo;
    int adat;
    Rang rang;
} Sor;

void beszuras(Sor ** sor, int adat, Rang rang) {
    Sor * uj = (Sor*) malloc(sizeof (Sor));
    uj->adat = adat;
    uj->rang = rang;
    uj->kovetkezo = NULL;
    if (*sor == NULL) {
 *sor = uj;
        return;
    }
    Sor * temp = *sor;
    Sor * prev = NULL;
    while (temp != NULL && temp->rang <= rang) {
        prev = temp;
        temp = temp->kovetkezo;
    }
    uj->kovetkezo = temp;
    if (prev == NULL) {
 *sor = uj;
    } else {
        prev->kovetkezo = uj;
    }
    //uj->kovetkezo = temp->kovetkezo;
    //temp->kovetkezo = uj;
}

int kovetkezo(Sor ** sor) {
    if (*sor == NULL) {
        return 0;
    }
    int adat = (*sor)->adat;
    Sor * kovetkezo = (*sor)->kovetkezo;
    free(*sor);
 *sor = kovetkezo;
    return adat;
}

void megjelenit(Sor * sor) {
    while (sor != NULL) {
        printf("Azonosito: %d\trang: ", sor->adat);
        switch (sor->rang) {
            case PROFESSZOR:
```

```
                printf("professzor\n");
                break;
            case DOCENS:
                printf("docens\n");
                break;
            case ADJUNKTUS:
                printf("adjunktus\n");
                break;
            case TANARSEGED:
                printf("tanarseged\n");
        }
        sor = sor->kovetkezo;
    }
}

int main() {
    Sor * sor = NULL;
    int menu;
    do {
        printf("Beszuras:  1\n");
        printf("Kivesz:    2\n");
        printf("Kilepes:   3\n");
        scanf("%d", &menu);
        if (menu == 1) {
            char kod[20];
            int azonosito;
            printf("Kerem a dokumentum azonositojat: ");
            scanf("%d", &azonosito);
            printf("Szemelyi kod: ");
            scanf("%s", kod);
            // P: professzr
            // D: docens
            // A: adjunktus
            // T: tanarseged
            switch (kod[0]) {
                case 'P':
                    beszuras(&sor, azonosito, PROFESSZOR);
                    break;
                case 'D':
                    beszuras(&sor, azonosito, DOCENS);
                    break;
                case 'A':
                    beszuras(&sor, azonosito, ADJUNKTUS);
                    break;
                case 'T':
                    beszuras(&sor, azonosito, TANARSEGED);
                    break;
                default:
                    printf("Hibas azonosito!\n");
            }
            if (sor != NULL && rand() % 100 >= 55) {
                printf("Elkeszult egy nyomtatas, azonositoja: %d\n\n",
kovetkezo(&sor));
            }
            megjelenit(sor);
        }
        if (menu == 2) {
            if (sor != NULL) {
                printf("A kovetkezo azonosito: %d\n", kovetkezo(&sor));
```

```
            } else {
                printf("Nincs tobb dokumentum!\n");
            }
        }
    } while (menu != 3);
    while (sor != NULL) {
        kovetkezo(&sor);
    }
    return EXIT_SUCCESS;
}
```

5.7.1.

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

struct row {
    int value;
    struct row *next;
    struct row *prev;
    struct row *end;
    struct row *first;
};

typedef struct row row;

void NewElement(int, row*);
int ElementOut(row*);

int main() {
    //két sor lesz, az elsőben tároljuk az aktuális sorát a Pascal
háromszögnek, a másodikban az összes sorát tároljuk
    row row1, row2;
    int max_level, current_level;
    //segédváltozók
    int i, k, x, y;

    row1.first = NULL;
    row1.end = NULL;
    row1.prev = NULL;
    row1.next = NULL;
    row2.first = NULL;
    row2.end = NULL;
    row2.prev = NULL;
    row2.next = NULL;

    printf("How many level do you want to crate in the Pascal's
triangle?: ");
    scanf("%d", &max_level);
printf("Representing the triangle is nice only before the 13(th)
row.\n");

    for (i = 0; i < max_level; i++) {
        for (current_level = 0; current_level <= i; current_level++) {
            if (current_level == 0) {
                NewElement(1, &row1);
                NewElement(1, &row2);
            }
            else {
                if (current_level != i) {
```

```
                    x = ElementOut(&row1);
                    y = row1.end->value;
                    NewElement(x + y, &row1);
                    NewElement(x + y, &row2);
                }
                else {
                    x = ElementOut(&row1);
                    NewElement(x, &row1);
                    NewElement(x, &row2);
                }
            }
        }
    }

    for (i = 0; i <= max_level; i++) {
        for (k = max_level; k > i; --k)
            printf("  ");
        for (current_level = 0; current_level < i; current_level++) {
            printf("%3d ", ElementOut(&row2));
        }
        printf("\n");
    }
    return 0;
}

void NewElement(int new_value, row *s) {
    row* actual;

    //első elem behelyezése a sorba
    if (s->first == NULL) {
        actual = (row*) malloc(sizeof (row));
        if (!actual)
            printf("The actual element wasn't created\n");
        actual->next = NULL;
        actual->prev = NULL;
        actual->value = new_value;
        s->first = actual;
        s->end = actual;
    }
    //további elemek elhelyezése a sorban
    else {
        actual = (row*) malloc(sizeof (row));
        if (!actual)
            printf("The actual element wasn't created\n");
        actual->value = new_value;
        actual->next = s->first;
        actual->prev = NULL;
        s->first->prev = actual;
        s->first = actual;
    }
}
int ElementOut(row *s) {
    row *temp;
    int value = 0;

    if (s->end != NULL) {
        value = s->end->value;
        if (s->end->prev != NULL) {
            temp = s->end->prev;
```

```
        }
        else {
            temp = NULL;
            s->first = NULL;
        }
        free(s->end);
        s->end = temp;
    }
    return value;
}
```

5.8.1.

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

struct row {
    char value[20];
    struct row* next;
    struct row* prev;
};

void UploadFIFO(char*, char*, struct row**, struct row**);
void InsertElement(char*, struct row**, struct row**);
void List_Check_Remove(char*, struct row**);
char* Check(char*, char*);

int main() {
    struct row* first = NULL, *end = NULL;
    char signal[2000] =
"abcdefg#abcdefg#abfdgcdefg#abdfcdefg#abcdfdefg#abcdefg#adfcvbcdefg#abcde
fg#abcdefg#aega";
    char pattern[20] = "abcdefg";

    printf("Recived signal:\n%s\n", signal);
    printf("\nChecking recived signal, the sequence pattern is: %s\n",
pattern);
    printf("Recived expressions between ## tags:\n");

    UploadFIFO(pattern, signal, &first, &end);
    List_Check_Remove(pattern, &first);
    return 0;
}

void UploadFIFO(char* pattern, char* signal, struct row** first, struct
row** end) {
    int i = 0, k;
    char temp[20];

    //csak akkor kezdi el az aktuális ciklust, ha még nincs vége az
adásnak
    while (signal[i] != '\0') {
        //ne rakjuk bele a mintákba a '#' karaktert
        if (signal[i] != '#') {
            temp[0] = signal[i];
            //ha esetleg egy szekvencia közben lenne vége az adásnak
            for (k = 1; (signal[i + k] != '#'); ++k) {
                if (signal[i + k] == '\0')
                    break;
```

```
                temp[k] = signal[i + k];
            }
            //lezárjuk a stringet
            temp[k] = '\0';
            InsertElement(temp, first, end);
            i = i + k;
        }
        ++i;
    }
}
void InsertElement(char* temp, struct row** first, struct row **end) {
    if (!(*first)) {
        (*first) = (struct row*)malloc(sizeof (struct row));
        (*end) = (*first);
        (*first)->prev = NULL;
        (*first)->next = NULL;
        strcpy((*first)->value, temp);
    }
    else {
        (*end)->next = (struct row*)malloc(sizeof (struct row));
        strcpy((*end)->next->value, temp);
        (*end)->next->next = NULL;
        (*end)->next->prev = (*end);
        (*end) = (*end)->next;
    }
}
void List_Check_Remove(char* pattern, struct row** first) {
    int i = 1;
    struct row* actual = (*first);

    while (*first) {
        printf("%4d. %s-> %s\n",i++, (*first)->value, Check(pattern,
(*first)->value));
        actual = (*first);
        (*first) = (*first)->next;
        free(actual);
    }
}
char* Check(char* pattern, char* temp) {
    if (strcmp(pattern, temp))
        return "Wrong";
    else
        return "OK";
}
```

5.9.1.

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

struct stack {
    int index;
    struct stack* next;
    struct stack* prev;
};
int CheckExpression(char*,struct stack**);
void InsertElement(struct stack**, int);
void RemoveElement(struct stack**, int);
```

```
int main() {
    int check;
    struct stack* end = NULL;
    char expression[100];

    printf("Please enter an expression e.g: (a(b(c((d))#))\n");
    scanf("%s", expression);
    check = CheckExpression(expression, &end);
    if (check > 0)
        printf("Wrong syntax!!! Too few close pharentheses!!!\n");
    else if (check < 0)
        printf("Wrong syntax!!! Cannot be read correctly, too many close
pharentheses!!!\n");
    else
        printf("Everything's all right, syntax OK.\n");
    return 0;
}

int CheckExpression(char* expr, struct stack** end) {
    int i, i_parentheses = 0;

    for (i = 0; i < strlen(expr); ++i) {
        if (expr[i] == '(') {
            InsertElement(end, i_parentheses++);
        }
        else if (expr[i] == ')') {
            if (i_parentheses == 0)
                return -1;
            else
                RemoveElement(end, i_parentheses--);
        }
    }
    return i_parentheses;
}
void InsertElement(struct stack** end, int index) {
    if (!(*end)) {
        (*end) = (struct stack*)malloc(sizeof (struct stack));
        (*end)->next = NULL;
        (*end)->prev = NULL;
        (*end)->index = index;
    }
    else {
        (*end)->next = (struct stack*)malloc(sizeof (struct stack));
        (*end)->next->prev = (*end);
        (*end) = (*end)->next;
        (*end)->next = NULL;
        (*end)->index = index;
    }
    printf("%d ->\n", (*end)->index);
}
void RemoveElement(struct stack** end, int index) {
    if (*end) {
        printf("<- %d\n", (*end)->index);
        if ((*end)->prev) {
            (*end) = (*end)->prev;
            free((*end)->next);
            (*end)->next = NULL;
        }
        else
```

```
            free(*end);
    }
}
```

6.1.1.

```
#include <stdio.h>
struct nodeType {
    int data;
    nodeType *left, *right, *parent;
};
typedef struct nodeType node;

void printNode(node* act) {
    printf("Act node: data: %d. ", act->data);
    printf("You can go left: %d, ", act->left?1:0 );
    printf("right: %d, ", act->right?1:0 );
    printf("parent: %d\n", act->parent?1:0 );
}

int select() {
    int result;
    printf("\nLeft - 1\n");
    printf("Right - 2\n");
    printf("Back - 3\n");
    printf("Quick - 4\n");
    printf("Your choice: ");
    scanf("%d", &result);
    return result;
}

int main() {
    node a={2, NULL, NULL, NULL}, b={11, NULL, NULL, NULL}, c={23, NULL,
NULL, NULL}, d={13, NULL, NULL, NULL},
    e={9, NULL, NULL, NULL}, f={7, NULL, NULL, NULL}, g={66, NULL, NULL,
NULL}, h={8, NULL, NULL, NULL},
    *root=&a, *act=root;
    int choice;

    a.left = &b;
    a.right = &c;
    b.left = &d;
    b.right = &e;
    b.parent = &a;
    d.right = &g;
    d.parent = &b;
    g.parent = &d;
    c.right = &f;
    c.parent = &a;
    f.left = &h;
    h.parent = &f;

    do {
        printNode(act);
        choice = select();
        switch (choice) {
            case 1:
                if (act->left) act = act->left;
                break;
            case 2:
```

```
                if (act->right) act = act->right;
                break;
            case 3:
                if (act->parent) act = act->parent;
                break;
        }
    } while (choice != 4);
    return 0;
}
```

6.1.2.

```
#include <stdio.h>
struct nodeType {
    int data;
    float data2;
    char string[50];
    struct nodeType *left, *right, *parent;
};
typedef struct nodeType node;

void printNode(node* act) {
    printf("Act node - data(int): %d, data(float): %f, data(string):
%s\n",
     act->data,act->data2,act->string);
    printf("You can go left: %d, ", act->left?1:0 );
    printf("right: %d, ", act->right?1:0 );
    printf("parent: %d\n", act->parent?1:0 );
}

void editNode(node* act) {
  printf("editing node\n");
  printf("data (int): ");
  scanf("%d",&(act->data));
  printf("data (float): ");
  scanf("%f",&(act->data2));
  printf("data (string): ");
  scanf("%s",act->string);
}

int select() {
    int result;
    printf("\nLeft - 1\n");
    printf("Right - 2\n");
    printf("Back - 3\n");
    printf("Edit - 4\n");
    printf("Quit - 5\n");
    printf("Your choice: ");
    scanf("%d", &result);
    return result;
}

int main() {
    node a={2, 0.2, "", NULL, NULL, NULL}, b={11, 0.11, "", NULL, NULL,
NULL},
    c={23, 0.23, "", NULL, NULL, NULL}, d={13, 0.13, "", NULL, NULL,
NULL},
    e={9, 0.9, "", NULL, NULL, NULL}, f={7, 0.7, "", NULL, NULL, NULL},
    g={66, 0.66, "", NULL, NULL, NULL}, h={8, 0.8, "", NULL, NULL, NULL},
    *root=&a, *act=root;
```

```
    int choice;

    a.left = &b;
    a.right = &c;
    b.left = &d;
    b.right = &e;
    b.parent = &a;
    d.right = &g;
    d.parent = &b;
    g.parent = &d;
    c.right = &f;
    c.parent = &a;
    f.left = &h;
    h.parent = &f;

    do {
        printNode(act);
        choice = select();
        switch (choice) {
            case 1:
                if (act->left) act = act->left;
                break;
            case 2:
                if (act->right) act = act->right;
                break;
            case 3:
                if (act->parent) act = act->parent;
                break;
          case 4:
            editNode(act);
            break;
        }
    } while (choice != 5);
    return 0;
}
```

6.2.1.

```
#include <stdio.h>
struct nodeType {
    int data;
    nodeType *left, *right;
};
typedef struct nodeType node;

void preOrder(node* act) {
    if (!act) return;
    printf("%d ", act->data);
    preOrder(act->left);
    preOrder(act->right);
}

void inOrder(node* act) {
    if (!act) return;
    inOrder(act->left);
    printf("%d ", act->data);
    inOrder(act->right);
}

void postOrder(node* act) {
```

```
    if (!act) return;
    postOrder(act->left);
    postOrder(act->right);
    printf("%d ", act->data);
}

int main() {
    node a={1, NULL, NULL}, b={2, NULL, NULL}, c={3, NULL, NULL}, d={4,
NULL, NULL},
    e={5, NULL, NULL}, f={7, NULL, NULL}, g={8, NULL, NULL}, h={9, NULL,
NULL},
    *root=&b;

    b.left = &f;
    b.right = &c;
    f.left = &d;
    f.right = &h;
    d.left = &a;
    d.right = &e;
    h.right = &g;
    printf("\nPre order: ");
    preOrder(root);
    printf("\nIn order: ");
    inOrder(root);
    printf("\nPost order: ");
    postOrder(root);

    return 0;
}
```

6.3.1.

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

struct nodeType {
    int price;
    char manufacturer[50];
    unsigned int infra : 1;
    float range;
    struct nodeType *less, *bigger;
};
typedef struct nodeType node;

void printNode(node* act) {
    printf("\n(price: %d, ", act->price);
    printf("manufacturer: %s, ", act->manufacturer);
    printf("infra: %d, ", act->infra);
    printf("range: %3.1f) ", act->range);
}

void printTree(node* act) {
    if (!act) return;
    printTree(act->less);
    printNode(act);
    printTree(act->bigger);
}
```

```
void freeTree(node* act) {
    if (!act) return;
    freeTree(act->less);
    freeTree(act->bigger);
    free(act);
}

void insertNode(node* sentinel, node* origNode) {
    node *act = sentinel, *newNode;
    newNode = (node*)malloc(sizeof(node));
    *newNode = *origNode;
    newNode->less = NULL;
    newNode->bigger = NULL;
    // because the sentinel we don't have to check insert as root
    while (1) {
        if (newNode->price <= act->price) { // if sentinel holds max key,
then equal elements must go to the left (sentinel can have only one
child)
            if (act->less == NULL) {
                act->less = newNode;
                return;
            } else act = act->less;
        } else {
            if (act->bigger == NULL) {
                act->bigger = newNode;
                return;
            }
            else act = act->bigger;
        }
    } // while end
}

int main() {
    node sentinel={LONG_MAX, "", 0, 0, NULL, NULL}, myNode={25, "ibm", 0,
5.2, NULL, NULL};
    insertNode(&sentinel, &myNode);
    printf("\nOrdered node list: ");
    myNode.price = 63;
    strcpy(myNode.manufacturer, "ms");
    myNode.infra=1;
    myNode.range=2.1;
    insertNode(&sentinel, &myNode);
    myNode.price = 11;
    strcpy(myNode.manufacturer, "genius");
    myNode.infra=1; myNode.range=4.5;
    insertNode(&sentinel, &myNode);
    myNode.price = 22;
    strcpy(myNode.manufacturer, "takamaka");
    myNode.infra=0;
    myNode.range=0.9;
    insertNode(&sentinel, &myNode);
    printTree(sentinel.less); // sentinel must not be printed
    freeTree(sentinel.less); // sentinel must not be deleted
    return 0;
}
```

6.3.2.

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

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

typedef struct nodeType {
    int price;
    char manufacturer[50];
    unsigned int infra : 1;
    float range;
    struct nodeType *less, *bigger, *parent;
} node;

void printNode(node* act) {
    printf("(price: %d, ", act->price);
    printf("manufacturer: %s, ", act->manufacturer);
    printf("infra: %d, ", act->infra);
    printf("range: %3.1f)\n", act->range);
}

void printTree(node* act) {
    if (!act) return;
    printTree(act->less);
    printNode(act);
    printTree(act->bigger);
}

void freeTree(node* act) {
    if (!act) return;
    freeTree(act->less);
    freeTree(act->bigger);
    free(act);
}

void insertNode(node* root, node* orig) {
  node* act=root;
  node* step=root;
  while (step!=NULL) {
    if (act->price >= orig->price) {
      step=(act->less);
    } else {
      step=(act->bigger);
    }
    if (step!=NULL) { act=step; }
  }
  if (act->price>=orig->price) {
    act->less=orig;
  } else { act->bigger=orig; }
  orig->parent=act;
}

int main() {
  node first = {25, "ibm", 0, 5.2, NULL, NULL};
  node myNode1 = {0,"",0,0.0,NULL,NULL};
  node myNode2 = {0,"",0,0.0,NULL,NULL};
  node myNode3 = {0,"",0,0.0,NULL,NULL};
  node *root = &first;
    myNode1.price = 63;
    strcpy(myNode1.manufacturer, "ms");
    myNode1.infra=1;
    myNode1.range=2.1;
```

```
    insertNode(root, &myNode1);
    myNode2.price = 11;
    strcpy(myNode2.manufacturer, "genius");
    myNode2.infra=1; myNode2.range=4.5;
    insertNode(root, &myNode2);
    myNode3.price = 22;
    strcpy(myNode3.manufacturer, "takamaka");
    myNode3.infra=0;
    myNode3.range=0.9;
    insertNode(root, &myNode3);
    printf("Ordered node list:\n");
    printTree(root); // sentinel must not be printed
    return 0;
}
```

6.4.1.

```
struct TPerson
{
   char Name[NAMELENGTH];
   int Born, Died;
   int ChildrenNum;
   struct TPerson * Children;
};

void ReadPerson(FILE * fd, struct TPerson * Head)
{
     int i;
     fscanf(fd, "%s", Head->Name);
     fscanf(fd, "%d %d %d", &Head->Born, &Head->Died, &Head-
>ChildrenNum);
     Head->Children = (struct TPerson*)malloc( sizeof(struct TPerson) *
Head->ChildrenNum);
     for (i = 0; i < Head->ChildrenNum; i++)
     {
         ReadPerson(fd, &Head->Children[i]);
     }
}

int PrintChildren(struct TPerson * Head, char * name, int year)
{
     int i;
     int exist = FALSE;
     if ( strcmp(Head->Name, name) == 0 )
     {
       exist = TRUE;
       if (Head->ChildrenNum > 0)
       {
          for (i = 0; i < Head->ChildrenNum; i++)
          {
              if (Head->Children[i].Died != -1)
              {
                 printf("%s, lived for ", Head->Children[i].Name);
                 printf("%d years\n", Head->Children[i].Died - Head-
>Children[i].Born);
              } else
              {
                 printf("%s, ", Head->Children[i].Name);
                 printf("%d years old\n", year - Head->Children[i].Born);
              }
```

```
          }
        } else printf("%s hasn't got children!\n", name);

      } else
      {
        for (i = 0; i < Head->ChildrenNum; i++)
        {
            if (!exist)
               exist = PrintChildren(&Head->Children[i], name, year);
            else
                PrintChildren(&Head->Children[i], name, year);
        }
      }
      return exist;
}

void FreeFamilyTree(struct TPerson * Head)
{
   int i;
   for (i = 0; i < Head->ChildrenNum; i++)
   {
       FreeFamilyTree( &Head->Children[i] );
   }
   free(Head->Children);
   Head->Children = NULL;
}

int main(int argc, char *argv[])
{
  struct TPerson Head;
  char Name[NAMELENGTH];
  FILE * fd = fopen(INPUT_FILE, "r");
  if (fd==NULL)
  {
     perror("Hiba");
     return 0;
  }
  ReadPerson(fd, &Head);
  fclose(fd);

  printf("Name: ");
  scanf("%s", Name);
  printf("Children of %s:\n*********************************\n", Name);
    if ( !PrintChildren(&Head, Name, ACTUAL_YEAR ) )
     printf("There is no person with name %s!\n", Name);

  FreeFamilyTree(&Head);
  return 0;
}
```

6.5.1.

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

#define INPUT_FILE "expr1.txt"
#define MAXLENGTH 15
#define ELEMENTNUM(a) (( a ) * 2 - 1)
```

```
typedef char TElement[MAXLENGTH];

struct TExpression
{
  int operandsnum;
  TElement * expr;
};

void PrintExpression(struct TExpression * E)
{
  int i;
  for (i = 0; i < ELEMENTNUM(E->operandsnum); i++)
    printf("%s ", E->expr[i]);
}

void MulDiv(struct TExpression * E)
{
  int i, j;
  int offs;
  double a;
  for (i = 0; i < ELEMENTNUM(E->operandsnum); i++)
  {
    offs = 0;
    if (strcmp(E->expr[i], "*") == 0)
    {
      a = atof(E->expr[i - 1]);
      a *= atof(E->expr[i + 1]);
      sprintf(E->expr[i - 1], "%g", a);
      offs = 1;
    }
    if (strcmp(E->expr[i], "/") == 0)
    {
      a = atof(E->expr[i - 1]);
      a /= atof(E->expr[i + 1]);
      sprintf(E->expr[i - 1], "%g", a);
      offs = 1;
    }
    if (offs)
    {
      for (j = i; j < (ELEMENTNUM(E->operandsnum) - 2); j++)
        strcpy(E->expr[j], E->expr[j + 2]);
      E->operandsnum--;
      i--;
    }
  }
}

void AddSub(struct TExpression * E)
{
  int i, j;
  int offs;
  double a;
  for (i = 0; i < ELEMENTNUM(E->operandsnum); i++)
  {
    offs = 0;
    if (strcmp(E->expr[i], "+") == 0)
    {
      a = atof(E->expr[i - 1]);
      a += atof(E->expr[i + 1]);
```

```
      sprintf(E->expr[i - 1], "%g", a);
      offs = 1;
    }
    if (strcmp(E->expr[i], "-") == 0)
    {
      a = atof(E->expr[i - 1]);
      a -= atof(E->expr[i + 1]);
      sprintf(E->expr[i - 1], "%g", a);
      offs = 1;
    }
    if (offs)
    {
      for (j = i; j < (ELEMENTNUM(E->operandsnum) - 2); j++)
        strcpy(E->expr[j], E->expr[j + 2]);
      E->operandsnum--;
      i--;
    }
  }
}

double GetValue(struct TExpression * E)
{

  MulDiv(E);
  AddSub(E);
  return atof(E->expr[0]);
}

void ReadExpression(FILE * fd, struct TExpression * E)
{
  int i;
  fscanf(fd, "%d", &E->operandsnum);
  E->expr = (TElement *)malloc(sizeof(TElement) * ELEMENTNUM(E-
>operandsnum));
  for (i = 0; i < ELEMENTNUM(E->operandsnum) ; i++)
  {
    fscanf(fd, "%s", *(E->expr + i));
  }
}

void FreeExpression(struct TExpression * E)
{
  free(E->expr);
  E->expr = NULL;
  E->operandsnum = 0;
}

int main(int argc, char * argv[])
{
  struct TExpression Expr;
  FILE * fd = fopen(argc > 1 ? argv[1] : INPUT_FILE, "r");
  if (fd == NULL)
  {
    perror("Error");
    return 1;
  }
  ReadExpression(fd, &Expr);
  fclose(fd);
  PrintExpression(&Expr);
```

```
  printf(" = %g\n", GetValue(&Expr));
  FreeExpression(&Expr);
  return 0;
}
```

6.6.1.-6.6.3.

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

struct binary_tree {
    int number;
    struct binary_tree* left;
    struct binary_tree* right;
};

void Insert(struct binary_tree*, int);
void List(struct binary_tree*);
int CountNode(struct binary_tree*);
int CountLeaf(struct binary_tree*);
int MaxDepth(struct binary_tree*);
int MaxKey(struct binary_tree*);
int MinDepth(struct binary_tree*);
int SameTree(struct binary_tree*, struct binary_tree*);

int main() {
    struct binary_tree* head = NULL, *head2 = NULL;
    int i;
    int tree_1[] = {7, 9, 13, 15, 45, 2, 1, 5, 94};
    int tree_2[] = {7, 9, 13, 15, 45, 2, 1, 5, 94};

    for (i = 0; i < sizeof(tree_1)/sizeof(int); ++i) {
        //első elem elhelyezése a fában
        if (!head) {
            head = (struct binary_tree*)malloc(sizeof(struct
binary_tree));
            head->left = NULL;
            head->right = NULL;
            head->number = tree_1[i];
        }
        //többi elem elhelyezése a fában
        else {
            Insert(head, tree_1[i]);
        }
    }
    for (i = 0; i < sizeof(tree_2)/sizeof(int); ++i) {
        //első elem elhelyezése a fában
        if (!head2) {
            head2 = (struct binary_tree*)malloc(sizeof(struct
binary_tree));
            head2->left = NULL;
            head2->right = NULL;
            head2->number = tree_2[i];
        }
        //többi elem elhelyezése a fában
        else {
            Insert(head2, tree_2[i]);
        }
    }
```

```
    printf("First tree:\n");
        List(head);
    printf("\n\nSecond tree:\n");
        List(head2);
    printf("\n\nThere are %d node in the first binary tree\n",
CountNode(head));
    printf("There are %d leaf in the first binary tree\n",
CountLeaf(head));
    printf("The max depth in the first tree is %d\n", MaxDepth(head));
    printf("The max key in the first tree is: %d\n", MaxKey(head));
    printf("The minimum depth in the first tree is: %d\n",
MinDepth(head));
    if(SameTree(head, head2))
        printf("The trees have the same shape\n");
    else
        printf("The trees don't have the same shape\n");
    return 0;
}

void Insert(struct binary_tree* head, int temp) {
    //amennyiben már létezik ilyen elem a fában, nem helyezzük el
mégegyszer
    if (temp == (head->number))
        ;
    else if (temp < head->number) {
        if (head->left)
            Insert(head->left, temp);
        else {
            head->left = (struct binary_tree*)malloc(sizeof(struct
binary_tree));
            head->left->number = temp;
            head->left->left = NULL;
            head->left->right = NULL;
        }
    }
    else if (temp > head->number) {
        if (head->right)
            Insert(head->right, temp);
        else {
            head->right = (struct binary_tree*)malloc(sizeof(struct
binary_tree));
            head->right->number = temp;
            head->right->left = NULL;
            head->right->right = NULL;
        }
    }
}
void List(struct binary_tree* head) {
    if (head->left)
        List(head->left);
    printf("%d ", head->number);
    if (head->right)
        List(head->right);
}
int CountNode(struct binary_tree* head) {
    if (head == NULL) {
        return 0;
    }
    else {
```

```
        return(CountNode(head->left) + 1 + CountNode(head->right));
  }
}
int CountLeaf(struct binary_tree* head) {
    if(head == NULL)
        return 0;
    //csak akkor tér vissza a függvény valós értékkel, ha az adott
csúcsnak nincsenek gyerekei
    if((head->left == NULL) && (head->right == NULL))
        return 1;
    else
        return (CountLeaf(head->left) + CountLeaf(head->right));
}
int MaxDepth(struct binary_tree* head) {
    int left_depth, right_depth;

    if (head == NULL)
        return 0;
    else {
        //mindegyik részfának meghatározza a mélységét
        left_depth = MaxDepth(head->left);
        right_depth = MaxDepth(head->right);
        //annak a részfának használjuk a mélységét, amelyik nagyobb
        if (left_depth > right_depth)
            return(left_depth + 1);
        else
            return(right_depth + 1);
    }
}
//a fa rendeése miatt, a legnagyobb elem nem más, mint a jobb alsó elem
int MaxKey(struct binary_tree* head) {
    if (head)
        while(head->right)
            head = head->right;
    return head->number;

}
//ugyanaz mint a MaxDepth, csak azt a részfát használjuk, amelyik kisebb
int MinDepth(struct binary_tree* head) {
    int left_depth, right_depth;

    if (head == NULL)
        return 0;
    else {
        left_depth = MaxDepth(head->left);
        right_depth = MaxDepth(head->right);
        if (left_depth < right_depth)
            return(left_depth + 1);
        else
            return(right_depth + 1);
    }
}
int SameTree(struct binary_tree* a, struct binary_tree* b) {
    //megnézzünk minden csúcsót, hogy ugyanannyi gyereke van-e
    if (a == NULL && b == NULL)
        return 1;
    else if (a != NULL && b != NULL) {
        return(
          SameTree(a->left, b->left) &&
```

```
          SameTree(a->right, b->right)
        );
    }
    else
        return 0;
}
```

7.1.1.

```
#include <stdio.h>

#define GRAPHSIZE 2048
#define INFINITY GRAPHSIZE*GRAPHSIZE
#define MAX(a, b) ((a > b) ? (a) : (b))

int e; // #non-zeros
int n; // #nodes
long dist[GRAPHSIZE][GRAPHSIZE];
long d[GRAPHSIZE]; // distances from the 0th node

void printD() {
    int i;
    printf("The shortest path form node 0 to node n\nnode n:  ");
    for (i = 1; i <= n; ++i)
        printf("%10d", i);
    printf("\ndistance: ");
    for (i = 1; i <= n; ++i) {
        printf("%10ld", d[i]);
    }
    printf("\n");
}

void dijkstra(int s) {
    int i, k, mini;
    int visited[GRAPHSIZE];

    for (i = 1; i <= n; ++i) {
        d[i] = INFINITY;
        visited[i] = 0;
    }

    d[s] = 0;

    for (k = 1; k <= n; ++k) {
        mini = -1;
        for (i = 1; i <= n; ++i)
            if (!visited[i] && ((mini == -1) || (d[i] < d[mini])))
                mini = i;
        visited[mini] = 1;
        for (i = 1; i <= n; ++i)
            if (dist[mini][i])
                if (d[mini] + dist[mini][i] < d[i])
                    d[i] = d[mini] + dist[mini][i];
    }
}

int main(int argc, char *argv[]) {
    int i, j;
    int u, v, w;
```

```
    FILE *fin = fopen("dist.txt", "r");
    fscanf(fin, "%d", &e);
    fscanf(fin, "%d", &n);
    for (i = 0; i < e; ++i)
        for (j = 0; j < e; ++j)
            dist[i][j] = 0;
    n = -1;
    for (i = 0; i < e; ++i) {
        fscanf(fin, "%d%d%d", &u, &v, &w);
        dist[u][v] = w;
        n = MAX(u, MAX(v, n));
    }
    fclose(fin);
    dijkstra(1);
    printD();
    return 0;
}

// dist.txt
6 5
1 2 3
1 3 2
2 4 6
2 5 3
3 5 7
5 4 2
```

7.2.1.-7.2.3.

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

typedef struct Graf {
    int csomopontok;
    double ** matrix;
} Graf;

typedef struct El {
    int pont1, pont2;
    double suly;
} El;

void grafInit(Graf * graf, int meret) {
    graf->csomopontok = meret;
    graf->matrix = NULL;
    if (meret > 0) {
        int sor, oszlop;
        graf->matrix = (double**) malloc(sizeof (double*) * graf-
>csomopontok);
        for (sor = 0; sor < graf->csomopontok; sor++) {
            graf->matrix[sor] = (double*) malloc(sizeof (double) * graf-
>csomopontok);
            for (oszlop = 0; oszlop < graf->csomopontok; oszlop++) {
                graf->matrix[sor][oszlop] = -1;
            }
        }
    }
}

void grafBetolt(const char * file, Graf * graf) {
```

```
    FILE * fd = fopen(file, "r");
    if (fd == NULL) {
        perror("Hiba");
        return;
    }
    int sor, oszlop;
    fscanf(fd, "%d", &graf->csomopontok);
    graf->matrix = (double**) malloc(sizeof (double*) * graf-
>csomopontok);
    for (sor = 0; sor < graf->csomopontok; sor++) {
        graf->matrix[sor] = (double*) malloc(sizeof (double) * graf-
>csomopontok);
        for (oszlop = 0; oszlop < graf->csomopontok; oszlop++) {
            fscanf(fd, "%lg", &graf->matrix[sor][oszlop]);
        }
    }
    fclose(fd);
}

void grafFelszabadit(Graf * graf) {
    int sor;
    for (sor = 0; sor < graf->csomopontok; sor++) {
        free(graf->matrix[sor]);
    }
    free(graf->matrix);
    graf->csomopontok = 0;
    graf->matrix = 0;
}

double dijkstra(int start, int veg, Graf graf, int * ut) {
    if (start == veg) {
        return 0.0;
    }
    double vegtelen = 1.0 / 0.0;
    int * kesz = (int*) malloc(sizeof (int) * graf.csomopontok);
    int * elozo = (int*) malloc(sizeof (int) * graf.csomopontok);
    double * d = (double*) malloc(sizeof (double) * graf.csomopontok);

    kesz[start] = 1;
    int i;
    for (i = 0; i < graf.csomopontok; i++) {
        kesz[i] = 0;
        d[i] = vegtelen;
        elozo[i] = -1;
        if (graf.matrix[start][i] > -1) {
            d[i] = graf.matrix[start][i]; // a start szomszedjainak
tavolsagara tudunk mar becslest adni
            elozo[i] = start; // az i-be a startbol jutunk el
        }
    }
    // mehet a kereses
    for (i = 0; i < graf.csomopontok; i++) {
        // keresunk egy csucsot, ami nincs bent a kesz-ben, es a tavolsag
ra minimalis
        int node = 0;
        // keresek olyan csomopontot ami meg nincs keszen
        while (node < graf.csomopontok && kesz[node])
            node++;
        // ha nem talaltunk ilyent, akkor vegeztunk
```

```
        if (node < graf.csomopontok) {

            int j;
            // ha talaltunk, akkor megkerssuk a legkisebb tavolsagut
            for (j = 0; j < graf.csomopontok; j++)
                if (d[j] < d[node] && !kesz[j])
                    node = j;

            kesz[node] = 1; // ezzel vegeztunk
            for (j = 0; j < graf.csomopontok; j++) {
                if (!kesz[j] && graf.matrix[node][j] > 0 &&
                        d[j] > d[node] + graf.matrix[node][j]) {
                    d[j] = d[node] + graf.matrix[node][j];
                    elozo[j] = node;
                }
            }
        } else {
            i = graf.csomopontok;
        }
    }

    int utIndex = graf.csomopontok;
    if (d[veg] < vegtelen) {

        int node = veg;
        while (node != start) {
            ut[ --utIndex ] = node;
            node = elozo[node];
        }
        ut[ --utIndex ] = node;
    }
    int ut_i;
    for (ut_i = utIndex; ut_i < graf.csomopontok - 1; ut_i++)
        printf("%d -- %lg -- ", ut[ut_i], graf.matrix[ut[ut_i]][ut[ut_i +
1]]);
    printf("%d\n", ut[ut_i]);

    double res = d[veg];
    free(elozo);
    free(d);
    free(kesz);
    return res;
}

int bejar(Graf graf, int start, int * szinek, int szin) {

    int i;
    int * meglatogatott = (int*) malloc(sizeof (int) * graf.csomopontok);
    for (i = 0; i < graf.csomopontok; i++) {
        meglatogatott[i] = 0;
    }
    int * nyitott = (int*) malloc(sizeof (int) * graf.csomopontok);
    int nyitottCsomopontok = 0;

    nyitott[0] = start;
    nyitottCsomopontok++;
    int node = start;
    int bejart = 0;
    while (nyitottCsomopontok > 0) {
```

```
        node = nyitott[ --nyitottCsomopontok ];
        if (meglatogatott[node] == 0) {
            meglatogatott[node] = 1;
            bejart++;
            if (szinek != NULL) {
                szinek[node] = szin;
            }
        }
        for (i = 0; i < graf.csomopontok; i++) {
            if (meglatogatott[i] == 0 && graf.matrix[node][i] > -1) {
                int j = 0;
                while (j < nyitottCsomopontok && nyitott[j] != i) {
                    j++;
                }
                if (j >= nyitottCsomopontok) {
                    nyitott[ nyitottCsomopontok++ ] = i;
                }
            }
        }
    }
    free(meglatogatott);
    free(nyitott);
    return bejart;
}

double kruskal(Graf graf, Graf mft) {
    // elek megszamlalasa
    int elekSzama = 0;
    int i, j;
    for (i = 0; i < graf.csomopontok; i++) {
        for (j = i + 1; j < graf.csomopontok; j++) {
            if (graf.matrix[i][j] > -1) {
                elekSzama++;
            }
        }
    }
    printf("%d el van a grafban\n", elekSzama);

    // elek kigyujtese
    El * elek = (El*) malloc(sizeof (El) * elekSzama);
    elekSzama = 0;
    for (i = 0; i < graf.csomopontok; i++) {
        for (j = i + 1; j < graf.csomopontok; j++) {
            if (graf.matrix[i][j] > -1) {
                elek[ elekSzama ].pont1 = i;
                elek[ elekSzama ].pont2 = j;
                elek[ elekSzama ].suly = graf.matrix[i][j];
                elekSzama++;
            }
        }
    }
    // elek rendezese
    for (i = 0; i < elekSzama - 1; i++) {
        int minIndex = i;
        for (j = i + 1; j < elekSzama; j++) {
            if (elek[ j ].suly < elek[ minIndex ].suly) {
                minIndex = j;
            }
        }
```

```
        El temp = elek[ i ];
        elek[ i ] = elek[ minIndex ];
        elek[ minIndex ] = temp;
    }

    // elek beillesztese
    int lefedve = 0;
    int * szinek = (int*)malloc(sizeof(int) * graf.csomopontok);
    for (i = 0; i < graf.csomopontok; i++) {
        szinek[i] = i;
    }
    int elIndex = 0;
    double osszSuly = 0.0;
    while  (elIndex < elekSzama) { //(lefedve < graf.csomopontok) {
        El el = elek[ elIndex ];
        if (szinek[el.pont1] != szinek[el.pont2]) {
            mft.matrix[ el.pont1 ][ el.pont2 ] = el.suly;
            osszSuly += el.suly;
            int szinRef = szinek[el.pont1];
            int szin2 = szinek[el.pont2];
            for (j = 0; j < graf.csomopontok; j++) {
                if (szinek[j] == szinRef) {
                    szinek[j] = szin2;
                }
            }
            // egyforma szinure szinezzuk oket
        }
        elIndex++;
    }
    free(elek);
    free(szinek);
    return osszSuly;
}

int main() {
    Graf graf;
    grafInit(&graf, 0);
    grafBetolt("terkep.txt", &graf);
    if (graf.csomopontok == 0) {
        return EXIT_FAILURE;
    }
    int * ut = (int*) malloc(graf.csomopontok * sizeof (int));
    int start, veg;
    printf("Kezdopont: ");
    scanf("%d", &start);
    printf("Cel: ");
    scanf("%d", &veg);
    double hossz = dijkstra(start, veg, graf, ut);
    printf("A legrovidebb ut hossza: %lg\n", hossz);

    int elerheto = bejar(graf, start, NULL, 0);
    printf("A %d. pontbol %d pont erheto el\n", start, elerheto);

    int * szinek = (int*) malloc(sizeof (int) * graf.csomopontok);
    int i;
    for (i = 0; i < graf.csomopontok; i++) {
        szinek[i] = 0;
    }
    int pont;
```

```
    int komponensek = 0;
    do {
        pont = 0;
        while (pont < graf.csomopontok && szinek[pont] > 0) {
            pont++;
        }
        if (pont < graf.csomopontok) {
            komponensek++;
            elerheto = bejar(graf, pont, szinek, komponensek);
            printf("A %d. komponensben %d csomopont van\n", komponensek,
elerheto);
        }
    } while (pont < graf.csomopontok);

    grafFelszabadit(&graf);
    free(ut);
    free(szinek);
    return EXIT_SUCCESS;
}

// terkep.txt
6
-1 16 13 -1 -1 -1
-1 -1 10 12 -1 -1
-1 4 -1 -1 14 -1
-1 -1 9 -1 -1 20
-1 -1 -1 7 -1 4
-1 -1 -1 -1 -1 -1
```

7.3.1.

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

typedef struct Graf {
    int csomopontok;
    double ** matrix;
} Graf;

typedef struct El {
    int pont1, pont2;
    double suly;
} El;

void grafInit(Graf * graf, int meret) {
    graf->csomopontok = meret;
    graf->matrix = NULL;
    if (meret > 0) {
        int sor, oszlop;
        graf->matrix = (double**) malloc(sizeof (double*) * graf-
>csomopontok);
        for (sor = 0; sor < graf->csomopontok; sor++) {
            graf->matrix[sor] = (double*) malloc(sizeof (double) * graf-
>csomopontok);
            for (oszlop = 0; oszlop < graf->csomopontok; oszlop++) {
                graf->matrix[sor][oszlop] = -1;
            }
        }
    }
```

```
}

void grafBetolt(const char * file, Graf * graf) {
    FILE * fd = fopen(file, "r");
    if (fd == NULL) {
        perror("Hiba");
        return;
    }
    int sor, oszlop;
    fscanf(fd, "%d", &graf->csomopontok);
    graf->matrix = (double**) malloc(sizeof (double*) * graf-
>csomopontok);
    for (sor = 0; sor < graf->csomopontok; sor++) {
        graf->matrix[sor] = (double*) malloc(sizeof (double) * graf-
>csomopontok);
        for (oszlop = 0; oszlop < graf->csomopontok; oszlop++) {
            fscanf(fd, "%lg", &graf->matrix[sor][oszlop]);
        }
    }
    fclose(fd);
}

void grafFelszabadit(Graf * graf) {
    int sor;
    for (sor = 0; sor < graf->csomopontok; sor++) {
        free(graf->matrix[sor]);
    }
    free(graf->matrix);
    graf->csomopontok = 0;
    graf->matrix = 0;
}

double kruskal(Graf graf, Graf mft) {
    // elek megszamlalasa
    int elekSzama = 0;
    int i, j;
    for (i = 0; i < graf.csomopontok; i++) {
        for (j = i + 1; j < graf.csomopontok; j++) {
            if (graf.matrix[i][j] > -1) {
                elekSzama++;
            }
        }
    }
    printf("%d el van a grafban\n", elekSzama);

    // elek kigyujtese
    El * elek = (El*) malloc(sizeof (El) * elekSzama);
    elekSzama = 0;
    for (i = 0; i < graf.csomopontok; i++) {
        for (j = i + 1; j < graf.csomopontok; j++) {
            if (graf.matrix[i][j] > -1) {
                elek[ elekSzama ].pont1 = i;
                elek[ elekSzama ].pont2 = j;
                elek[ elekSzama ].suly = graf.matrix[i][j];
                elekSzama++;
            }
        }
    }
    // elek rendezese
```

```
    for (i = 0; i < elekSzama - 1; i++) {
        int minIndex = i;
        for (j = i + 1; j < elekSzama; j++) {
            if (elek[ j ].suly < elek[ minIndex ].suly) {
                minIndex = j;
            }
        }
        El temp = elek[ i ];
        elek[ i ] = elek[ minIndex ];
        elek[ minIndex ] = temp;
    }

    // elek beillesztese
    int lefedve = 0;
    int * szinek = (int*) malloc(sizeof (int) * graf.csomopontok);
    for (i = 0; i < graf.csomopontok; i++) {
        szinek[i] = i;
    }
    int elIndex = 0;
    double osszSuly = 0.0;
    while (elIndex < elekSzama) { //(lefedve < graf.csomopontok) {
        El el = elek[ elIndex ];
        if (szinek[el.pont1] != szinek[el.pont2]) {
            mft.matrix[ el.pont1 ][ el.pont2 ] = el.suly;
            mft.matrix[ el.pont2 ][ el.pont1 ] = el.suly;
            osszSuly += el.suly;
            int szinRef = szinek[el.pont1];
            int szin2 = szinek[el.pont2];
            for (j = 0; j < graf.csomopontok; j++) {
                if (szinek[j] == szinRef) {
                    szinek[j] = szin2;
                }
            }
        }
        elIndex++;
    }
    free(elek);
    free(szinek);
    return osszSuly;
}

int main() {
    Graf graf;
    grafInit(&graf, 0);
    grafBetolt("kruskal.txt", &graf);
    if (graf.csomopontok == 0) {
        return EXIT_FAILURE;
    }
    Graf minfeszfa;
    grafInit(&minfeszfa, graf.csomopontok);

    double minFeszFaHossz = kruskal(graf, minfeszfa);
    printf("A minimalis feszitofa ossz sulya: %lg\n", minFeszFaHossz);
    int i, j;
    for (i = 0; i < minfeszfa.csomopontok; i++) {
        for (j = i + 1; j < minfeszfa.csomopontok; j++) {
            if (minfeszfa.matrix[i][j] > -1) {
                printf("%d -- %d, suly: %lg\n", i, j,
minfeszfa.matrix[i][j]);
```

```
            }
        }
    }

    grafFelszabadit(&graf);
    grafFelszabadit(&minfeszfa);
    return EXIT_SUCCESS;
}

// kruskal.txt
11
-1  7 -1  5 -1 -1 -1 -1 -1 -1 -1
 7 -1  8  9  7 -1 -1  5 -1 -1 -1
-1  8 -1 -1  5 -1 -1 -1 -1 -1 -1
 5  9 -1 -1 15  6 -1 -1 -1 -1 -1
-1  7  5 15 -1  8  9 -1 -1 -1 -1
-1 -1 -1  6  8 -1 11  7 -1 -1 -1
-1 -1 -1 -1  9 11 -1 -1  5 -1 -1
-1  5 -1 -1  7 -1 -1 -1 -1  6 -1
-1 -1 -1 -1 -1 -1  5 -1 -1 -1 10
-1 -1 -1 -1 -1 -1 -1  6 -1 -1 -1
-1 -1 -1 -1 -1 -1 -1 -1 10 -1 -1
```

7.4.1.-7.4.3.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <set>
#include <algorithm>
#include <limits>

using namespace std;

const double vegtelen = numeric_limits<double>::infinity();

class FlowGraph {
public:
    FlowGraph();
    void Betolt(string FileNev);
    void Folyam(int pont1, int pont2, double f);
    double Folyam(int pont1, int pont2) const;
    double Maradek(int pont1, int pont2) const;
    bool Szomszedok(int pont1, int pont2) const;
    void ResetFolyam();
    double Ki(int pont) const;
    double Be(int pont) const;
    int Csomopontok() const;
private:
    vector< vector<double> > flow;
    vector< vector<double> > matrix;
    int n;

    void MatrixBetolt(istream & is, vector< vector<double> > & matrix,
int n);
    void Init(int mennyi);
};

FlowGraph::FlowGraph() {
    n = 0;
```

```
}

void FlowGraph::MatrixBetolt(istream & is, vector< vector<double> > &
matrix, int n) {
    matrix.clear();
    int i, j;
    for (i = 0; i < this->n; i++) {
        vector<double> sor;
        for (j = 0; j < this->n; j++) {
            double szam;
            is >> szam;
            if (szam == -1)
                szam = vegtelen;
            sor.push_back(szam);
        }
        matrix.push_back(sor);
    }
}

void FlowGraph::Init(int mennyi) {
    n = mennyi;
    matrix.resize(n);
    int i;
    for (i = 0; i < n; i++)
        matrix[i].resize(n, vegtelen);
}

void FlowGraph::Folyam(int pont1, int pont2, double f) {
    flow[pont1][pont2] = f;
    flow[pont2][pont1] = -f;
}

void FlowGraph::ResetFolyam() {
    int i, j;
    for (i = 0; i < flow.size(); i++) {
        for (j = 0; j < flow.size(); j++) {
            flow[i][j] = 0;
        }
    }
}

void FlowGraph::Betolt(string FileNev) {
    ifstream ifs(FileNev.c_str());
    if (ifs) {
        ifs >> n;
        MatrixBetolt(ifs, matrix, n);
        ifs.close();
    } else cout << "hibas megnyitas" << endl;
    int i, j;
    flow.resize(n);
    for (i = 0; i < n; i++) {
        flow[i].resize(n, 0);
        for (j = 0; j < n; j++)
            if (matrix[i][j] == vegtelen)
                matrix[i][j] = -vegtelen;
    }
}

double FlowGraph::Ki(int pont) const {
```

```
    int i;
    double res = 0;
    for (i = 0; i < n; i++)
        if (flow[pont][i] > 0)
            res += flow[pont][i];
    return res;
}

double FlowGraph::Be(int pont) const {
    int i;
    double res = 0;
    for (i = 0; i < n; i++)
        if (flow[i][pont] > 0)
            res += flow[i][pont];
    return res;
}

bool FlowGraph::Szomszedok(int pont1, int pont2) const {
    return Maradek(pont1, pont2) > 0.0;
}

double FlowGraph::Folyam(int pont1, int pont2) const {
    return flow[pont1][pont2];
}

double FlowGraph::Maradek(int pont1, int pont2) const {
    return matrix[pont1][pont2] - flow[pont1][pont2];
}

int FlowGraph::Csomopontok() const {
    return n;
}

bool bejar(int pont1, int pont2, FlowGraph & graf, vector<int> & ut) {
    int i;
    set<int> visited;
    vector<int> openset;
    vector<int> elozo;
    elozo.resize(graf.Csomopontok(), -1);
    openset.push_back(pont1);
    int node = pont1;
    while (!openset.empty() && node != pont2) {
        node = openset.back();
        openset.pop_back();
        visited.insert(node);
        for (i = 0; i < graf.Csomopontok(); i++) {
            if (visited.find(i) == visited.end() && graf.Szomszedok(node,
i)
                    && find(openset.begin(), openset.end(), i) ==
openset.end()) {
                openset.push_back(i);
                elozo[i] = node;
            }
        }
    }
    vector<int>::iterator iter = elozo.begin();
    ut.clear();
    if (node == pont2) {
        int node = pont2;
```

```
        while (node != pont1) {
            ut.insert(ut.begin(), node);
            node = elozo[node];
        }
        ut.insert(ut.begin(), node);
    }
    return node == pont2;
}

double FordFulkerson(int start, int end, FlowGraph & flow) {
    vector<int> ut;
    do {
        bejar(start, end, flow, ut);
        if (ut.size() > 0) {
            double min = flow.Maradek(ut[0], ut[1]);
            int i;
            for (i = 1; i < ut.size() - 1; i++)
                if (flow.Maradek(ut[i], ut[i + 1]) < min)
                    min = flow.Maradek(ut[i], ut[i + 1]);

            for (i = 0; i < ut.size() - 1; i++) {
                flow.Folyam(ut[i], ut[i + 1], min + flow.Folyam(ut[i],
ut[i + 1]));
            }
        }
    } while (ut.size() > 0);
    return flow.Be(end);
}

void legtobbSorHova(FlowGraph & graf, int honnan) {
    int maxIndex = 0;
    double maxSor = 0.0;
    int index;
    for (index = 0; index < graf.Csomopontok(); index++) {
        if (index != honnan) {
            graf.ResetFolyam();
            double mennyiseg = FordFulkerson(honnan, index, graf);
            if (mennyiseg > maxSor) {
                maxSor = mennyiseg;
                maxIndex = index;
            }
        }
    }
    cout << honnan << "-bol a legtobb sor " << maxIndex << "-be
szallithato, ";
    cout << "a mennyiseg: " << maxSor << endl;
}

void komponensek(FlowGraph & graf) {
    vector<bool> megneztuk;
    megneztuk.resize(graf.Csomopontok(), false);
    int keres;
    int komponensek = 0;
    int maxKomponens = 0;
    do {
        keres = 0;
        while (keres < megneztuk.size() && megneztuk[keres])
            keres++;
```

```
        if (keres < megneztuk.size()) {
            komponensek++;
            int pont1 = keres;
            int i;
            set<int> visited;
            vector<int> openset;

            openset.push_back(pont1);
            int node = pont1;
            while (!openset.empty()) {
                node = openset.back();
                openset.pop_back();

                visited.insert(node);
                megneztuk[node] = true;

                for (i = 0; i < graf.Csomopontok(); i++) {
                    if (visited.find(i) == visited.end() &&
graf.Szomszedok(node, i)
                            && find(openset.begin(), openset.end(), i) ==
openset.end()) {
                        openset.push_back(i);
                    }
                }
                vector<int>::iterator iter = openset.begin();
            }
            if (maxKomponens < visited.size()) {
                maxKomponens = visited.size();
            }
        }

    } while (keres < megneztuk.size());
    cout << "A legnagyobb komponens " << maxKomponens << " varosbol all"
<< endl;
}

int main(int argc, char** argv) {
    FlowGraph graf;
    graf.Betolt("pipes.txt");
    int sorgyar, cel;
    cout << "Hol legyen a sorgyar? " << endl;
    cin >> sorgyar;
    cout << "Hol legyen a cel? " << endl;
    cin >> cel;

    double max = FordFulkerson(sorgyar, cel, graf);
    cout << sorgyar << " es " << cel << " kozott max " << max << "
mennyisegu sor szallithato" << endl;

    legtobbSorHova(graf, sorgyar);
    graf.ResetFolyam();
    komponensek(graf);
    return 0;
}


// pipes.txt
6
-1 16 13 -1 -1 -1
```

```
-1 -1 10 12 -1 -1
-1 4 -1 -1 14 -1
-1 -1 9 -1 -1 20
-1 -1 -1 7 -1 4
-1 -1 -1 -1 -1 -1
```

7.5.1.

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

#define SIZE 7

struct connected_nodes {
    int connected_node;
    struct connected_nodes* next;
};
struct nodes {
    int node;
    int clas;
    int visited;
    struct connected_nodes* edges;
    struct nodes* next;
};
struct list {
    struct nodes* actual;
    struct list* next;
};
void CreateGraph(int, int**, struct nodes*);
int BFS(struct nodes*);
void CreateList(struct nodes*, struct list**, struct list**);
void FreeList(struct list**);

int main() {
    int i, j;
    struct nodes my_node;
    int** dynamic_matrix = NULL;
    int adjacenci_matrix[SIZE][SIZE] = {
        {0, 0, 0, 1, 1, 1, 0},
        {0, 0, 0, 1, 1, 0, 0},
        {0, 0, 0, 0, 1, 0, 1},
        {1, 1, 0, 0, 0, 0, 0},
        {1, 1, 1, 0, 0, 0, 0},
        {1, 0, 0, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0}
    };

    //létrehozunk egy dinamikus tömböt, hogy át tudjuk adni azt egy
függvénynek paraméterként
    dynamic_matrix = (int**)malloc(SIZE*sizeof(int*));
    for (i = 0; i < SIZE; ++i) {
        dynamic_matrix[i] = (int*)malloc(SIZE*sizeof(int));
    }
    for(i = 0; i < SIZE; ++i)
        for (j = 0; j < SIZE; ++j)
            dynamic_matrix[i][j] = adjacenci_matrix[i][j];

    my_node.edges = NULL;
    my_node.next = NULL;
```

```
    my_node.node = 0;
    CreateGraph(SIZE, dynamic_matrix, &my_node);
    if (BFS(&my_node)) {
        printf("\nThe graph is not bipartite!\n");
    }
    else {
        printf("\nThe nodes have been divided into two disjoint sets.\n");
        printf("The graph is bipartite.\n");
    }
    for (i = 0; i < SIZE; ++i) {
        free(dynamic_matrix[i]);
    }
    free(dynamic_matrix);
    return 0;
}

void CreateGraph(int size, int** adjacenci_matrix, struct nodes* my_node)
{
    int i = 0, j = 0;
    struct connected_nodes* temp_ptr;
    my_node->node = 0;
    my_node->visited = 0;
    my_node->clas = 0;
    //az első csúcs szomszédaink meghatározása
    for(i = 0; i < size; ++i) {
        if(adjacenci_matrix[0][i]) {
            //többi szomszéd
            if (my_node->edges) {
                temp_ptr->next = (struct connected_nodes*)malloc(sizeof
(struct connected_nodes));
                temp_ptr = temp_ptr->next;
                temp_ptr->connected_node = i;
                temp_ptr->next = NULL;
            }
            //első szomszéd
            else {
                my_node->edges = (struct connected_nodes*)malloc(sizeof
(struct connected_nodes));
                temp_ptr = my_node->edges;
                temp_ptr->connected_node = i;
                temp_ptr->next = NULL;
            }
        }
    }
    //többi csúcs szomszédainak keresése
    for(i = 1; i < size; ++i) {
        my_node->next = (struct nodes*)malloc(sizeof (struct nodes));
        my_node = my_node->next;
        my_node->node = i;
        my_node->next = NULL;
        my_node->edges = NULL;
        my_node->visited = 0;
        my_node->clas = 0;
        for(j = 0; j < size; ++j) {
            if(adjacenci_matrix[i][j]) {
                if (my_node->edges) {
                    temp_ptr->next = (struct connected_nodes*)malloc(sizeof
(struct connected_nodes));
                    temp_ptr = temp_ptr->next;
```

```
                    temp_ptr->connected_node = j;
                    temp_ptr->next = NULL;
                }
                else {
                    my_node->edges = (struct connected_nodes*)malloc(sizeof
(struct connected_nodes));
                    temp_ptr = my_node->edges;
                    temp_ptr->connected_node = j;
                    temp_ptr->next = NULL;
                }
            }
        }
    }
}
int BFS(struct nodes* root) {
    int current_node;
    struct connected_nodes* cn_temp;
    struct nodes* n_temp;
    struct list* list_temp, *list_temp1, * list_temp2,
*list_temp_for_print;

    //két listát fogok készíteni, mindig az első listából generáljuk a
következőt,
    //az első lista tartalmazza az adott szint csúcsait, a második listát
így ebből fogjuk generálni,
    //amint kész van, átállítjuk a mutatóját, hogy ez legyen az első
lista
    list_temp2 = NULL;
    root->clas = 1;
    root->visited = 1;
    //elkészítem az első listát, 1 elemmel
    printf("Creating first list (nodes in the current list): ");
    list_temp1 = (struct list*)malloc(sizeof(struct list));
    list_temp1->actual = root;
    list_temp1->next = NULL;
    list_temp_for_print = list_temp1;
    while (list_temp_for_print) {
        printf("%d ", list_temp_for_print->actual->node);
        list_temp_for_print = list_temp_for_print->next;
    }
    printf("\n");

    while (list_temp1) {
        list_temp = list_temp1;
        while (list_temp) {
            cn_temp = list_temp->actual->edges;
            //van-e szomszédja az adott csúcsnak
            while (cn_temp) {
                //melyik az?
                current_node = cn_temp->connected_node;
                n_temp = root;
                //ráállítjuk az n_temp pointert
                while (n_temp->node != current_node)
                    n_temp = n_temp->next;
                //megnézzük, hogy bejártuk-e már, ha nem, akkor de és
beállítjuk az ellentétes osztályt
                if (n_temp->visited == 0) {
                    //a 2 jelzi, hgoy az adott csúcsot meg fogjuk
látogatni, nehogy felfűzzünk a következő listába azt is ami már volt
```

```
                n_temp->visited = 2;
                n_temp->clas = (-1) * list_temp1->actual->clas;
            }
            //ha már bejártuk, akkor megvizsgáljuk, hogy egyeznek-e az
osztályaik
            else {
                if (list_temp1->actual->clas == n_temp->clas)
                    return 1;
                else {
                    ;
                }
            }
            //nézzük a következő szomszédot
            cn_temp = cn_temp->next;
        }
        list_temp = list_temp->next;
    }
    //készítjük a következő listát
    //ha elösször készítünk listát
    printf("Creating next list (nodes in the current list): ");
    CreateList(root, &list_temp1, &list_temp2);
    FreeList(&list_temp1);
    list_temp1 = list_temp2;
    list_temp2 = NULL;
    list_temp_for_print = list_temp1;
    while (list_temp_for_print) {
        printf("%d ", list_temp_for_print->actual->node);
        list_temp_for_print = list_temp_for_print->next;
    }
    printf("\n");
  }
  return 0;
}
void CreateList(struct nodes* root, struct list** ptr1, struct list**
ptr2) {
  struct nodes* n_temp;
  struct connected_nodes* connected_node_temp;
  struct list *first_list_temp, *second_list_temp;
  int current_node;

  first_list_temp = *ptr1;
  //ameddig létezik az első listának eleme, lépegetünk rajta
  while(first_list_temp) {
    connected_node_temp = first_list_temp->actual->edges;
    //ameddig egy elemének létezik szomszédja, azokat fűzzük fel a
második listába
    while (connected_node_temp) {
        //megkeressük, hogy ez a szomszéd melyik csúcs is valójában
        current_node = connected_node_temp->connected_node;
        n_temp = root;
        while (n_temp->node != current_node)
            n_temp = n_temp->next;
        //első elem elhelyezése, de csak azt amit még egyáltalán nem
jártunk be, erre kell a 2-es jelző
        if (!(*ptr2) && (n_temp->visited == 2)) {
            n_temp->visited = 1;
            (*ptr2) = (struct list*)malloc(sizeof(struct list));
            (*ptr2)->actual = n_temp;
            (*ptr2)->next = NULL;
```

```
                second_list_temp = *ptr2;
            }
            //többi elem elhelyezése,  de csak azt amit még egyáltalán nem
járt unk be
            else if(n_temp->visited == 2) {
                n_temp->visited = 1;
                second_list_temp->next = (struct
list*)malloc(sizeof(struct list));
                second_list_temp = second_list_temp->next;
                second_list_temp->actual = n_temp;
                second_list_temp->next = NULL;
            }
            connected_node_temp = connected_node_temp->next;
        }
        first_list_temp = first_list_temp->next;
    }
}
void FreeList(struct list** head) {
    struct list* temp1, *temp2;

    temp1 = (*head);
    while (temp1) {
        temp2 = temp1->next;
        free(temp1);
        temp1 = temp2;
    }
    *head = NULL;
}
```

7.6.1.

```
#include <stdio.h>

typedef struct {
    int u, v, weight;
}Edge;

#define INFINITY 10000
#define SIZE 5

void BellmanFord(int, int, int, int*, Edge*);
void PrintDistances(int, int[]);

int main() {
    Edge edges[1024];
    int i, j;
    int edges_num = 0, source_node = 2;
    int distances_from_source[SIZE];
    int adjacenci_matrix[SIZE][SIZE] = {
        {0, 6, 0, 7, 0},
        {0, 0, 5, 8, -4},
        {0, -2, 0, 0, 0},
        {0, 0, -3, 9, 0},
        {2, 0, 7, 0, 0}
    };
    printf("The adjacenci matrix:\n");

    for (i = 0; i < SIZE; ++i) {
        for (j = 0; j < SIZE; ++j) {
            printf("%d ", adjacenci_matrix[i][j]);
```

```
            if (adjacenci_matrix[i][j] != 0) {
                edges[edges_num].u = i;
                edges[edges_num].v = j;
                edges[edges_num].weight = adjacenci_matrix[i][j];
                edges_num++;
            }
        }
        printf("\n");
    }
    printf("\nRun Belmann-Ford algorithm to node %d\n", source_node);
    BellmanFord(source_node, SIZE, edges_num, distances_from_source,
edges);
    printf("\nMinimal distances from node %d:\n", source_node);
    PrintDistances(SIZE, distances_from_source);
    return 0;
}

void PrintDistances(int size, int* distances_from_source) {
    int i;

    for (i = 0; i < size; ++i)
        printf("to %d\t", i + 1);
    printf("\n");

    for (i = 0; i < size; ++i)
        printf("%d\t", distances_from_source[i]);

    printf("\n");
}

void BellmanFord(int source, int size, int edges_num, int*
distances_from_source, Edge* edges) {
    int i, j;

    for (i = 0; i < size; ++i)
        distances_from_source[i] = INFINITY;

    distances_from_source[source] = 0;

    for (i = 0; i < size - 1; ++i)
        for (j = 0; j < edges_num; ++j)
            if (distances_from_source[edges[j].u] + edges[j].weight <
distances_from_source[edges[j].v])
                distances_from_source[edges[j].v] =
distances_from_source[edges[j].u] + edges[j].weight;
}
```

7.6.2.

```
#include <stdio.h>

typedef struct {
    int u, v, weight;
}Edge;

#define INFINITY 10000
#define SIZE 8

int BellmanFord(int, int, int, int*, Edge*);
void PrintDistances(int, int[]);
```

```
int main() {
    Edge edges[1024];
    int i, j, max_distance = 0, temp_max_distance;
    int edges_num = 0;
    int distances_from_source[SIZE];
    int adjacenci_matrix[SIZE][SIZE] = {
        {0, 1, 1, 0, 0, 0, 0, 0},
        {1, 0, 0, 1, 0, 0, 0, 0},
        {1, 0, 0, 0, 1, 1, 0, 0},
        {0, 1, 0, 0, 0, 0, 1, 0},
        {0, 0, 1, 0, 0, 0, 0, 0},
        {0, 0, 1, 0, 0, 0, 0, 1},
        {0, 0, 0, 1, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 1, 0, 0},
    };
    printf("Adjacenci matrix:\n");
    for (i = 0; i < SIZE; ++i) {
        for (j = 0; j < SIZE; ++j) {
            printf("%d ", adjacenci_matrix[i][j]);
            if (adjacenci_matrix[i][j] != 0) {
                edges[edges_num].u = i;
                edges[edges_num].v = j;
                edges[edges_num].weight = adjacenci_matrix[i][j];
                edges_num++;
            }
        }
        printf("\n");
    }
    for (i = 0;i < SIZE; ++i) {
        temp_max_distance = BellmanFord(i, SIZE, edges_num,
distances_from_source, edges);
        PrintDistances(SIZE, distances_from_source);
        if (temp_max_distance > max_distance)
            max_distance = temp_max_distance;
        }
    printf("\nThe tree's diameter is: %d\n", max_distance);
    return 0;
}

void PrintDistances(int size, int* distances_from_source) {
    int i;

    printf("Distances from ");
    for (i = 0; i < size; ++i) {
        if (distances_from_source[i] == 0)
            printf("%d:\n", i+1);
    }
    for (i = 0; i < size; ++i)
        printf("to %d\t", i + 1);
    printf("\n");
    for (i = 0; i < size; ++i)
        printf("%d\t", distances_from_source[i]);
    printf("\n");
}

int BellmanFord(int source, int size, int edges_num, int*
distances_from_source, Edge* edges) {
    int i, j, max = 0;
```

```
    for (i = 0; i < size; ++i)
        distances_from_source[i] = INFINITY;
    distances_from_source[source] = 0;
    for (i = 0; i < size - 1; ++i) {
        for (j = 0; j < edges_num; ++j) {
            if (distances_from_source[edges[j].u] + edges[j].weight <
distances_from_source[edges[j].v]) {
                distances_from_source[edges[j].v] =
distances_from_source[edges[j].u] + edges[j].weight;
            }
        }
    }
    for (i = 0; i < size; ++i) {
        if (distances_from_source[i] > max)
            max = distances_from_source[i];
    }
    return max;
}
```

7.6.3.

```
#include <stdio.h>

typedef struct {
    int u, v;
    float weight;
}Edge;

#define INFINITY 10000
#define SIZE 3

void ModifiedBellmanFord(int, int, int, float*, Edge*);
void PrintMaxMoney(int, float[]);

int main() {
    Edge edges[1024];
    int i, j;
    int edges_num = 0, source_node = 0;
    float distances_from_source[SIZE];
    float adjacenci_matrix[SIZE][SIZE] = {
        {1, 2, 3},
        {1./2, 1, 1./4},
        {1./3, 4, 1}
    };
    printf("Adjacenci matrix for exchange table:\n");
    for (i = 0; i < SIZE; ++i) {
        for (j = 0; j < SIZE; ++j) {
            printf("%f           ", adjacenci_matrix[i][j]);
            if (adjacenci_matrix[i][j] != 0) {
                edges[edges_num].u = i;
                edges[edges_num].v = j;
                edges[edges_num].weight = adjacenci_matrix[i][j];
                edges_num++;
            }
        }
        printf("\n");
    }
    ModifiedBellmanFord(source_node, SIZE, edges_num,
distances_from_source, edges);
```

```
    PrintMaxMoney(SIZE, distances_from_source);
    return 0;
}

void PrintMaxMoney(int size, float* distances_from_source) {
    int i;

    printf("\nMax money at current currancys:\n");

    for (i = 0; i < size; ++i)
        printf("at %d\t\t", i + 1);
    printf("\n");
    for (i = 0; i < size; ++i)
        printf("%f\t", distances_from_source[i]);
    printf("\n");
}
void ModifiedBellmanFord(int source, int size, int edges_num, float*
distances_from_source, Edge* edges) {
    int i, j;


    for (i = 0; i < size; ++i)
        distances_from_source[i] = 0;
    distances_from_source[source] = 1;
    for (i = 0; i < size - 1; ++i) {
        for (j = 0; j < edges_num; ++j) {
            if (distances_from_source[edges[j].u] * edges[j].weight >
distances_from_source[edges[j].v]) {
                distances_from_source[edges[j].v] =
distances_from_source[edges[j].u] * edges[j].weight;
            }
        }
    }
}
```

7.7.1.

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

#define SIZE 5

struct edges {
    int current_node;
    struct edges* next;
};
struct node {
    int parent;
    int current_node;
    struct edges* current_node_edges;
    struct node* next;
};
struct edge_list {
    int u;
    int v;
    int weight;
    struct edge_list* next;
};

void Union(int, int, int, struct node*, struct node*);
```

```
int Where(int, int, int, struct node*, struct node*);
void CreateDisjunctGraphs(struct node*, int);
void FreeGraph(struct node*);
void FreeGraphList(struct node*);
void SortByWeight(struct edge_list**);
void FreeEdgeList(struct edge_list*);

int main() {
    int temp_num, i, j, k;
    //segédváltozók, egy él két végpontjához
    int source, destiny;
    //segédváltozók a HOLVAN művelet elvégzéséhez, ezekben tárolom, hogy
egy él két végpontja melyik gráfban van
    int destiny_disjunct_graph_root, source_disjunct_graph_root;
    //az a gráf amiben tárolom a még diszjunk gráfokat
    struct node my_disjunct_graph;
    //segédlista az HOLVAN művelet elvégzéséhez
    struct node nodes_in_row, *nodes_in_row_temp;
    //a hálozat éleinek tárolására szolgáló lista a rendezés előtt
    struct edge_list* my_edge_list = NULL;
    //segédlista a hálozat éleinek a rendezéséhez
    struct edge_list *edge_list_temp;
    //segédváltozó átmentei állapot kiíratásához
    struct edges* edges_temp;
    //mindig a rendezett lista elejére mutat
    struct edge_list* my_sorted_edge_list;
    int adjacenci_matrix[SIZE][SIZE] = {
        {0, 3, 1, 0, 9},
        {0, 4, 2, 4, 0},
        {0, 4, 0, 1, 7},
        {2, 0, 5, 0, 0},
        {1, 0, 3, 0, 4}
    };

    //hálozat beolvasása
    printf("The adjacenci matrix:\n");
    for (i = 0; i < SIZE; ++i) {
        for (k = 0; k < i; ++k)
               printf("0 ");
        for (j = i; j < SIZE; ++j) {
            printf("%d ", adjacenci_matrix[i][j]);
            if (adjacenci_matrix[i][j] != 0) {
                //első elem beszúrása egy listába, amit később csökkenő
sorrendbe rendezünk
                if (!my_edge_list) {
                    my_edge_list = (struct edge_list*)malloc(sizeof(struct
edge_list));
                    edge_list_temp = my_edge_list;
                    edge_list_temp->next = NULL;
                    edge_list_temp->u = i+1;
                    edge_list_temp->v = j+1;
                    edge_list_temp->weight = adjacenci_matrix[i][j];
                }
                //töbi elem beszúrása a listába
                else {
                    edge_list_temp->next = (struct
edge_list*)malloc(sizeof(struct edge_list));
                    edge_list_temp = edge_list_temp->next;
                    edge_list_temp->next = NULL;
```

```
                    edge_list_temp->u = i+1;
                    edge_list_temp->v = j+1;
                    edge_list_temp->weight = adjacenci_matrix[i][j];
                }
            }
        }
        printf("\n");
    }
    //az előbb elkészített lista rendezése
    SortByWeight(&my_edge_list);
    my_sorted_edge_list = my_edge_list;
    edge_list_temp = my_sorted_edge_list;
    my_edge_list = my_edge_list->next;
    while (my_edge_list) {
        //a segédlista elejére tesszük a legnagyobb elemet
        SortByWeight(&my_edge_list);
        //a rendezett lista következő eleme így mindig a következő
legnagyobb érték lesz, ami még megmaradt a segédlistában
        edge_list_temp->next = my_edge_list;
        edge_list_temp = edge_list_temp->next;
        //a segédlista első elemét kiszedem a segédlistából, a következő
legnagyobb elemet szeretném megtalálni a hátralevő elemek között
        my_edge_list = my_edge_list->next;
    }
    edge_list_temp = my_sorted_edge_list;
    printf("\nList of the links from the biggest:\n");
    while (edge_list_temp) {
        printf("Link (%d-%d), weight: %d\n", edge_list_temp->u,
edge_list_temp->v, edge_list_temp->weight );
        edge_list_temp = edge_list_temp->next;
    }
    //elkészítem a diszjunkt gráfokat, eleinte minden pont képez egy
külön gráfot
    CreateDisjunctGraphs(&my_disjunct_graph, SIZE);
    //létrehozom a segédlistát is az HOLVAN művelethez
    CreateDisjunctGraphs(&nodes_in_row, SIZE);

    //segédváltozóval végigmegyek a rendezett éllistámon
    edge_list_temp = my_sorted_edge_list;
    while (edge_list_temp) {
        printf("\nDisjunct graphs, before analyse link (%d-%d) with weight
%d:\n", edge_list_temp->u, edge_list_temp->v, edge_list_temp->weight);
        nodes_in_row_temp = &my_disjunct_graph;
        while (nodes_in_row_temp) {
            printf("Root nood: %d the link(s): ", nodes_in_row_temp-
>current_node);
            edges_temp = nodes_in_row_temp->current_node_edges;
            while(edges_temp) {
                printf("%d ", edges_temp->current_node);
                edges_temp = edges_temp->next;
            }
            printf("\n");
            nodes_in_row_temp = nodes_in_row_temp->next;
        }
        //az éppen aktuális él két végpontját eltárolom
        destiny = edge_list_temp->u;
        source = edge_list_temp->v;
        //megnézem, hogy melyik gráfokban vannak ezek az élek
```

```
        destiny_disjunct_graph_root = Where(0, 0, destiny,
&my_disjunct_graph, &nodes_in_row);
        source_disjunct_graph_root = Where(0, 0, source,
&my_disjunct_graph, &nodes_in_row);

        //a fügvény csak akkor működik, ha a cél gráf gyökere kissebb,
mint a másik gráfé
        if (destiny_disjunct_graph_root > source_disjunct_graph_root) {
            temp_num = destiny_disjunct_graph_root;
            destiny_disjunct_graph_root = source_disjunct_graph_root;
            source_disjunct_graph_root = temp_num;
        }
        //ha az él két vége nem ugyanabban a gráfban van, elvégezzük az
UNIÓ műveletet
        if (destiny_disjunct_graph_root != source_disjunct_graph_root) {
            Union(SIZE, destiny_disjunct_graph_root,
source_disjunct_graph_root, &my_disjunct_graph, &nodes_in_row);
        }
        else {
            printf("The nodes are in the same graph!\n");
        }
        //ha elkészült a feszítőfánk, kiírjuk a hálozat maximális
sávszélességét
        if (!my_disjunct_graph.next) {
            printf("\nPerfect, only one graph remain.\n");
            printf("The minimum spanning tree has been created, the last
link weight is: %d\n", edge_list_temp->weight);
            break;
        }
        edge_list_temp = edge_list_temp->next;
    }
    //ha még maradt legalább két egymástól diszjunk gráf, akkor a
sávszélesség 0 lesz
    if (my_disjunct_graph.next) {
        printf("\nSorry, more than one graph remain.\n");
        printf("Cannot create minimum spanning tree!\n");
    }

    FreeGraph(&my_disjunct_graph);
    FreeGraphList(&nodes_in_row);
    FreeEdgeList(my_sorted_edge_list);
    return 0;
}

void CreateDisjunctGraphs(struct node* my_node, int size) {
    int i;

    my_node->parent = 1;
    my_node->current_node = 1;
    my_node->current_node_edges = NULL;
    my_node->next = NULL;
    for(i = 2; i <= size; ++i) {
        my_node->next = (struct node*)malloc(sizeof(struct node));
        my_node = my_node->next;
        my_node->parent = i;
        my_node->current_node = i;
        my_node->current_node_edges = NULL;
        my_node->next = NULL;
    }
```

```
}
int Where(int recursion, int temp_parent, int u, struct node* actual,
struct node* original) {
    int i = 0;
    struct node* temp;
    struct edges* temp_edge;

    //az actual a diszjunk gráfokból képezett listán megy végig (a
gyökerein)
    while (actual) {
        //ha megtaláltuk a keresett csúcsot, visszatérési értéknek
megadjuk, melyik gráfba tartozik
        if (u == actual->current_node || temp_parent)
            return actual->parent;
        else {
            //ha az adott csúcsnak létezik gyereke, akkor meghívjuk rá a
HOLVAN függvényt
            temp_edge = actual->current_node_edges;
            while (temp_edge) {
                temp = original;
                //megkeressük a segédlistán az adott gyereket
                while(temp_edge->current_node != temp->current_node)
                    temp = temp->next;
                temp_parent = Where(1, temp_parent, u, temp, original);
                //ha már megtaláltuk a csúcsot a gráfban, ne nézzük tovább
a többi gyereket is, térjünk vissza a megfelelő értékkel, a rekurzió
miatt
                if (temp_parent)
                    return actual->parent;
                temp_edge = temp_edge->next;
            }
        }
        if (recursion)
            return 0;
        actual = actual->next;
    }
    return 0;
}
void Union(int size, int destiny, int source, struct node*
my_dijunct_graphs , struct node* original) {
    struct node* temp_node, *source_node, *source_node_prev,
*destiny_node;
    struct edges* temp_edge;

    source_node = my_dijunct_graphs;
    destiny_node = my_dijunct_graphs;
    //megkeressük, hol van a két gráf gyökere, amikre el szeretnénk
végezni az UNNIÓ műveletet
    while (source_node->current_node != source) {
        source_node_prev = source_node;
        source_node = source_node->next;
    }
    while (destiny_node->current_node != destiny)
        destiny_node = destiny_node->next;

    //ha a source és a destiny gráfok egymás után vannak a diszjunk
gráfokat tartalmazó listában
    //kivesszük a diszjunk gráfok listájából a source elemet
    if (destiny_node->next->current_node == source) {
```

```
        destiny_node->next = source_node->next;
        source_node->next = NULL;
        //átállítom a szülőjét, így a HOLVAN művelet visszatérési értéke
konzisztens lesz
        source_node->parent = destiny_node->parent;
        //megkeressük a source elemet az eredeti listában is
        temp_node = original;
        while(temp_node->current_node != source)
            temp_node = temp_node->next;
        //átállítjuk neki is a szűlőjét
        temp_node->parent = destiny_node->parent;

        //megnézzük, az adott csúcsnak vannak-e szomszédai
        //ha még nincs, megcsiináljuk az elsőt
        if (!destiny_node->current_node_edges) {
            destiny_node->current_node_edges = (struct
edges*)malloc(sizeof(struct edges));
            destiny_node->current_node_edges->next = NULL;
            destiny_node->current_node_edges->current_node = source_node-
>current_node;
            //az eredeti listánál is beállítjuk a destiny csúcs
szomszédait
            temp_node = original;
            while (temp_node->current_node != destiny_node->current_node)
                temp_node = temp_node->next;
            temp_node->current_node_edges = destiny_node-
>current_node_edges;
        }
        //ha van már szomszéd, akkor megkeressük a láncolt lista végét és
oda szúrjuk be az új elemet
        else {
            temp_edge = destiny_node->current_node_edges;
            while (temp_edge->next)
                temp_edge = temp_edge->next;
            temp_edge->next = (struct edges*)malloc(sizeof(struct edges));
            temp_edge = temp_edge->next;
            temp_edge->current_node = source;
            temp_edge->next = NULL;
        }
    }
    //ha nem egymás után vannak a listában a sorce és a destiny gráfok
    else {
        source_node_prev->next = source_node->next;
        source_node->next = NULL;
        source_node->parent = destiny_node->parent;
        temp_node = original;
        while(temp_node->current_node != source)
            temp_node = temp_node->next;
        temp_node->parent = destiny_node->parent;
        temp_edge = destiny_node->current_node_edges;
        if (!destiny_node->current_node_edges) {
            destiny_node->current_node_edges = (struct
edges*)malloc(sizeof(struct edges));
            destiny_node->current_node_edges->next = NULL;
            destiny_node->current_node_edges->current_node = source_node-
>current_node;
            temp_node = original;
            while (temp_node->current_node != destiny_node->current_node)
                temp_node = temp_node->next;
```

```
            temp_node->current_node_edges = destiny_node-
>current_node_edges;
        }
        else {
            temp_edge = destiny_node->current_node_edges;
            while (temp_edge->next)
                temp_edge = temp_edge->next;
            temp_edge->next = (struct edges*)malloc(sizeof(struct edges));
            temp_edge = temp_edge->next;
            temp_edge->current_node = source;
            temp_edge->next = NULL;
        }
    }
}
void SortByWeight(struct edge_list** root) {
    int min;
    struct edge_list* temp, *temp_prev, *min_weight, *min_weight_prev;

    //elösször beállítjuk, hogy a lista első eleme legyen a legnagyobb
    temp_prev = *root;
    temp = temp_prev->next;
    min_weight = *root;
    min_weight_prev = NULL;
    min = (*root)->weight;
    //végigmegyünk a listán és megkeressünk a tényleges legnagyobb elemet
    while (temp) {
        if (temp->weight < min) {
            min = temp->weight;
            min_weight = temp;
            min_weight_prev = temp_prev;
            temp_prev = temp;
            temp = temp->next;
        }
        else {
            temp_prev = temp;
            temp = temp->next;
        }
    }
    //mutatók átállítása, hogy a legnagyobb elem legyen elől
    //ha a lista első eleme a legnagyobb elem, akkor nem kell semmit se
csinálni
    if (!min_weight_prev)
        ;
    //ha a második a legnagyobb elem
    else if (min_weight_prev == (*root)) {
        (*root)->next = min_weight->next;
        min_weight->next = min_weight_prev;
        (*root) = min_weight;
    }
    //egyébként
    else {
        min_weight_prev->next = min_weight->next;
        min_weight->next = (*root);
        (*root) = min_weight;
    }
}
void FreeGraph(struct node* root) {
    struct edges* temp_edge1, *temp_edge2;
    struct node* temp_node1, *temp_node2;
```

```
    temp_edge1 = root->current_node_edges;
    while (temp_edge1) {
        temp_edge2 = temp_edge1;
        temp_edge1 = temp_edge1->next;
        free(temp_edge2);
    }

    temp_node1 = root->next;
    while (temp_node1) {
        temp_edge1 = temp_node1->current_node_edges;
        while (temp_edge1) {
            temp_edge2 = temp_edge1;
            temp_edge1 = temp_edge1->next;
            free(temp_edge2);
        }
        temp_node2 = temp_node1;
        temp_node1 = temp_node1->next;
        free(temp_node2);
    }
}
void FreeGraphList(struct node* root) {
    struct node* temp_node1, *temp_node2;

    temp_node1 = root->next;
    while (temp_node1) {
        temp_node2 = temp_node1;
        temp_node1 = temp_node1->next;
        free(temp_node2);
    }
}
void FreeEdgeList(struct edge_list* temp1) {
    struct edge_list* temp2;

    while(temp1) {
        temp2 = temp1;
        temp1 = temp1->next;
        free(temp2);
    }
}
```

7.7.2.

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

#define SIZE 5

struct edges {
    int current_node;
    struct edges* next;
};
struct node {
    int parent;
    int current_node;
    struct edges* current_node_edges;
    struct node* next;
};
struct edge_list {
    int u;
```

```
    int v;
    int weight;
    struct edge_list* next;
};

void Union(int, int, int, struct node*, struct node*);
int Where(int, int, int, struct node*, struct node*);
void CreateDisjunctGraphs(struct node*, int);
void FreeGraph(struct node*);
void FreeGraphList(struct node*);
void SortByWeight(struct edge_list**);
void FreeEdgeList(struct edge_list*);

int main() {
    int temp_num, i, j;
    //segédváltozók, egy él két végpontjához
    int source, destiny;
    //segédváltozók a HOLVAN művelet elvégzéséhez, ezekben tárolom, hogy
egy él két végpontja melyik gráfban van
    int destiny_disjunct_graph_root, source_disjunct_graph_root;
    //az a gráf amiben tárolom a még diszjunk gráfokat
    struct node my_disjunct_graph;
    //segédlista az HOLVAN művelet elvégzéséhez
    struct node nodes_in_row, *nodes_in_row_temp;
    //a hálozat éleinek tárolására szolgáló lista a rendezés előtt
    struct edge_list* my_edge_list = NULL;
    //segédlista a hálozat éleinek a rendezéséhez
    struct edge_list *edge_list_temp;
    //segédváltozó átmentei állapot kiíratásához
    struct edges* edges_temp;
    //mindig a rendezett lista elejére mutat
    struct edge_list* my_sorted_edge_list;
    int adjacenci_matrix[SIZE][SIZE] = {
        {0, 0, 5, 0, 3},
        {0, 6, 9, 0, 8},
        {0, 8, 0, 2, 0},
        {6, 2, 5, 0, 5},
        {5, 0, 5, 1, 0}
    };

    //hálozat beolvasása
    printf("The adjacenci matrix:\n");
    for (i = 0; i < SIZE; ++i) {
        for (j = 0; j < SIZE; ++j) {
            printf("%d ", adjacenci_matrix[i][j]);
            if (adjacenci_matrix[i][j] != 0) {
                //első elem beszúrása egy listába, amit később csökkenő
sorrendbe rendezünk
                if (!my_edge_list) {
                    my_edge_list = (struct edge_list*)malloc(sizeof(struct
edge_list));
                    edge_list_temp = my_edge_list;
                    edge_list_temp->next = NULL;
                    edge_list_temp->u = i+1;
                    edge_list_temp->v = j+1;
                    edge_list_temp->weight = adjacenci_matrix[i][j];
                }
                //többi elem beszúrása a listába
                else {
```

```
                    edge_list_temp->next = (struct
edge_list*)malloc(sizeof(struct edge_list));
                    edge_list_temp = edge_list_temp->next;
                    edge_list_temp->next = NULL;
                    edge_list_temp->u = i+1;
                    edge_list_temp->v = j+1;
                    edge_list_temp->weight = adjacenci_matrix[i][j];
                }
            }
        }
        printf("\n");
    }
    //az előbb elkészített lista rendezése
    SortByWeight(&my_edge_list);
    my_sorted_edge_list = my_edge_list;
    edge_list_temp = my_sorted_edge_list;
    my_edge_list = my_edge_list->next;
    while (my_edge_list) {
        //a segédlista elejére tesszük a legnagyobb elemet
        SortByWeight(&my_edge_list);
        //a rendezett lista következő eleme így mindig a következő
legnagyobb érték lesz, ami még megmaradt a segédlistában
        edge_list_temp->next = my_edge_list;
        edge_list_temp = edge_list_temp->next;
        //a segédlista első elemét kiszedem a segédlistából, a következő
legnagyobb elemet szeretném megtalálni a hátralevő elemek között
        my_edge_list = my_edge_list->next;
    }
    edge_list_temp = my_sorted_edge_list;
    printf("\nList of the links from the biggest:\n");
    while (edge_list_temp) {
        printf("Link (%d-%d), weight: %d\n", edge_list_temp->u,
edge_list_temp->v, edge_list_temp->weight );
        edge_list_temp = edge_list_temp->next;
    }
    //elkészítem a diszjunkt gráfokat, eleinte minden pont képez egy
külön gráfot
    CreateDisjunctGraphs(&my_disjunct_graph, SIZE);
    //létrehozom a segédlistát is az HOLVAN művelethez
    CreateDisjunctGraphs(&nodes_in_row, SIZE);

    //segédváltozóval végigmegyek a rendezett éllistámon
    edge_list_temp = my_sorted_edge_list;
    while (edge_list_temp) {
        printf("\nDisjunct graphs, before analyse link (%d-%d) with weight
%d:\n", edge_list_temp->u, edge_list_temp->v, edge_list_temp->weight);
        nodes_in_row_temp = &my_disjunct_graph;
        while (nodes_in_row_temp) {
            printf("Root nood: %d the link(s): ", nodes_in_row_temp-
>current_node);
            edges_temp = nodes_in_row_temp->current_node_edges;
            while(edges_temp) {
                printf("%d ", edges_temp->current_node);
                edges_temp = edges_temp->next;
            }
            printf("\n");
            nodes_in_row_temp = nodes_in_row_temp->next;
        }
        //az éppen aktuális él két végpontját eltárolom
```

```
        destiny = edge_list_temp->u;
        source = edge_list_temp->v;
        //megnézem, hogy melyik gráfokban vannak ezek az élek
        destiny_disjunct_graph_root = Where(0, 0, destiny,
&my_disjunct_graph, &nodes_in_row);
        source_disjunct_graph_root = Where(0, 0, source,
&my_disjunct_graph, &nodes_in_row);

        //a fügvény csak akkor működik, ha a cél gráf gyökere kissebb,
mint a másik gráfé
        if (destiny_disjunct_graph_root > source_disjunct_graph_root) {
            temp_num = destiny_disjunct_graph_root;
            destiny_disjunct_graph_root = source_disjunct_graph_root;
            source_disjunct_graph_root = temp_num;
        }
        //ha az él két vége nem ugyanabban a gráfban van, elvégezzük az
UNIÓ műveletet
        if (destiny_disjunct_graph_root != source_disjunct_graph_root) {
            Union(SIZE, destiny_disjunct_graph_root,
source_disjunct_graph_root, &my_disjunct_graph, &nodes_in_row);
        }
        else {
            printf("The nodes are in the same graph!\n");
        }
        //ha elkészült a feszítőfánk, kiírjuk a hálozat maximális
sávszélességét
        if (!my_disjunct_graph.next) {
            printf("\nPerfect, only one graph remain.\n");
            printf("The network's bandwidth: %d\n", edge_list_temp-
>weight);
            break;
        }
        edge_list_temp = edge_list_temp->next;
    }
    //ha még maradt legalább két egymástól diszjunk gráf, akkor a
sávszélesség 0 lesz
    if (my_disjunct_graph.next) {
        printf("\nSorry, more than one graph remains.\n");
        printf("The network's bandwidth: 0\n");
    }

    FreeGraph(&my_disjunct_graph);
    FreeGraphList(&nodes_in_row);
    FreeEdgeList(my_sorted_edge_list);
    return 0;
}

void CreateDisjunctGraphs(struct node* my_node, int size) {
    int i;

    my_node->parent = 1;
    my_node->current_node = 1;
    my_node->current_node_edges = NULL;
    my_node->next = NULL;
    for(i = 2; i <= size; ++i) {
        my_node->next = (struct node*)malloc(sizeof(struct node));
        my_node = my_node->next;
        my_node->parent = i;
        my_node->current_node = i;
```

```
        my_node->current_node_edges = NULL;
        my_node->next = NULL;
    }
}
int Where(int recursion, int temp_parent, int u, struct node* actual,
struct node* original) {
    struct node* temp;
    struct edges* temp_edge;

    //az actual a diszjunk gráfokból képezett listán megy végig (a
gyökerein)
    while (actual) {
        //ha megtaláltuk a keresett csúcsot, visszatérési értéknek
megadjuk, melyik gráfba tartozik
        if (u == actual->current_node || temp_parent)
            return actual->parent;
        else {
            //ha az adott csúcsnak létezik gyereke, akkor meghívjuk rá a
HOLVAN függvényt
            temp_edge = actual->current_node_edges;
            while (temp_edge) {
                temp = original;
                //megkeressük a segédlistán az adott gyereket
                while(temp_edge->current_node != temp->current_node)
                    temp = temp->next;
                temp_parent = Where(1, temp_parent, u, temp, original);
                //ha már megtaláltuk a csúcsot a gráfban, ne nézzük tovább
a többi gyereket is, térjünk vissza a megfelelő értékkel, a rekurzió
miatt
                if (temp_parent)
                    return actual->parent;
                temp_edge = temp_edge->next;
            }
        }
        if (recursion)
            return 0;
        actual = actual->next;
    }
    return 0;
}
void Union(int size, int destiny, int source, struct node*
my_dijunct_graphs , struct node* original) {
    struct node* temp_node, *source_node, *source_node_prev,
*destiny_node;
    struct edges* temp_edge;

    source_node = my_dijunct_graphs;
    destiny_node = my_dijunct_graphs;
    //megkeressük, hol van a két gráf gyökere, amikre el szeretnénk
végezni az UNIÓ műveletet
    while (source_node->current_node != source) {
        source_node_prev = source_node;
        source_node = source_node->next;
    }
    while (destiny_node->current_node != destiny)
        destiny_node = destiny_node->next;

    //ha a source és a destiny gráfok egymás után vannak a diszjunk
gráfokat tartalmazó listában
```

```
    //kivesszük a diszjunk gráfok listájából a source elemet
    if (destiny_node->next->current_node == source) {
        destiny_node->next = source_node->next;
        /*destiny_original->mod_next = source_original->next;*/
        source_node->next = NULL;
        /*source_original->mod_next = NULL;*/
        //átállítom a szülőjét, így a HOLVAN művelet visszatérési értéke
konzisztens lesz
        source_node->parent = destiny_node->parent;
        //megkeressük a source elemet az eredeti listában is
        temp_node = original;
        while(temp_node->current_node != source)
            temp_node = temp_node->next;
        //átállítjuk neki is a szűlőjét
        temp_node->parent = destiny_node->parent;

        //megnézzük, az adott csúcsnak vannak-e szomszédai
        //ha még nincs, megcsiináljuk az elsőt
        if (!destiny_node->current_node_edges) {
            destiny_node->current_node_edges = (struct
edges*)malloc(sizeof(struct edges));
            destiny_node->current_node_edges->next = NULL;
            destiny_node->current_node_edges->current_node = source_node-
>current_node;
            //az eredeti listánál is beállítjuk a destiny csúcs
szomszédait
            temp_node = original;
            while (temp_node->current_node != destiny_node->current_node)
                temp_node = temp_node->next;
            temp_node->current_node_edges = destiny_node-
>current_node_edges;
        }
        //ha van már szomszéd, akkor megkeressük a láncolt lista végét és
oda szúrjuk be az új elemet
        else {
            temp_edge = destiny_node->current_node_edges;
            while (temp_edge->next)
                temp_edge = temp_edge->next;
            temp_edge->next = (struct edges*)malloc(sizeof(struct edges));
            temp_edge = temp_edge->next;
            temp_edge->current_node = source;
            temp_edge->next = NULL;
        }
    }
    //ha nem egymás után vannak a listában a sorce és a destiny gráfok
    else {
        source_node_prev->next = source_node->next;
        source_node->next = NULL;
        source_node->parent = destiny_node->parent;
        temp_node = original;
        while(temp_node->current_node != source)
            temp_node = temp_node->next;
        temp_node->parent = destiny_node->parent;
        temp_edge = destiny_node->current_node_edges;
        if (!destiny_node->current_node_edges) {
            destiny_node->current_node_edges = (struct
edges*)malloc(sizeof(struct edges));
            destiny_node->current_node_edges->next = NULL;
```

```
            destiny_node->current_node_edges->current_node = source_node-
>current_node;
            temp_node = original;
            while (temp_node->current_node != destiny_node->current_node)
                temp_node = temp_node->next;
            temp_node->current_node_edges = destiny_node-
>current_node_edges;
        }
        else {
            temp_edge = destiny_node->current_node_edges;
            while (temp_edge->next)
                temp_edge = temp_edge->next;
            temp_edge->next = (struct edges*)malloc(sizeof(struct edges));
            temp_edge = temp_edge->next;
            temp_edge->current_node = source;
            temp_edge->next = NULL;
        }
    }
}
void SortByWeight(struct edge_list** root) {
    int max;
    struct edge_list* temp, *temp_prev, *max_weight, *max_weight_prev;

    //elösször beállítjuk, hogy a lista első eleme legyen a legnagyobb
    temp_prev = *root;
    temp = temp_prev->next;
    max_weight = *root;
    max_weight_prev = NULL;
    max = (*root)->weight;
    //végigmegyünk a listán és megkeressünk a tényleges legnagyobb elemet
    while (temp) {
        if (temp->weight > max) {
            max = temp->weight;
            max_weight = temp;
            max_weight_prev = temp_prev;
            temp_prev = temp;
            temp = temp->next;
        }
        else {
            temp_prev = temp;
            temp = temp->next;
        }
    }
    //mutatók átállítása, hogy a legnagyobb elem legyen elől
    //ha a lista első eleme a legnagyobb elem, akkor nem kell semmit se
csinálni
    if (!max_weight_prev)
        ;
    //ha a második a legnagyobb elem
    else if (max_weight_prev == (*root)) {
        (*root)->next = max_weight->next;
        max_weight->next = max_weight_prev;
        (*root) = max_weight;
    }
    //egyébként
    else {
        max_weight_prev->next = max_weight->next;
        max_weight->next = (*root);
        (*root) = max_weight;
```

```
    }
}
void FreeGraph(struct node* root) {
    struct edges* temp_edge1, *temp_edge2;
    struct node* temp_node1, *temp_node2;

    temp_edge1 = root->current_node_edges;
    while (temp_edge1) {
        temp_edge2 = temp_edge1;
        temp_edge1 = temp_edge1->next;
        free(temp_edge2);
    }

    temp_node1 = root->next;
    while (temp_node1) {
        temp_edge1 = temp_node1->current_node_edges;
        while (temp_edge1) {
            temp_edge2 = temp_edge1;
            temp_edge1 = temp_edge1->next;
            free(temp_edge2);
        }
        temp_node2 = temp_node1;
        temp_node1 = temp_node1->next;
        free(temp_node2);
    }
}
void FreeGraphList(struct node* root) {
    struct node* temp_node1, *temp_node2;

    temp_node1 = root->next;
    while (temp_node1) {
        temp_node2 = temp_node1;
        temp_node1 = temp_node1->next;
        free(temp_node2);
    }
}
void FreeEdgeList(struct edge_list* temp1) {
    struct edge_list* temp2;

    while(temp1) {
        temp2 = temp1;
        temp1 = temp1->next;
        free(temp2);
    }
}
```

8.1.1.

```
#include <stdio.h>

void print(double* tomb, int size) {
    int idxI;
    printf("[");
    for (idxI=0; idxI<size; idxI++)
        printf("%lf ", tomb[idxI]);
    printf("]");

}
int minIndex(double* tomb, int size) {
    int minIndex = 0;
```

```
    int idxI;
    for (idxI=1; idxI<size; idxI++)
        if (tomb[idxI] < tomb[minIndex])
            minIndex = idxI;
    return minIndex;
}
void swap(double* tomb, int id1, int id2) {
    double temp;
    temp=tomb[id1];
    tomb[id1]=tomb[id2];
    tomb[id2]=temp;
}
void minSort(double *tomb, int len) { // complexity: n*n, ascending order
    int idxI, c;

    for(idxI=0; idxI<len; idxI++) {
        c=minIndex(tomb+idxI, len-idxI); // min search at the end of the
array
        c=c+idxI;
        swap(tomb, c, idxI);
    }
}
void boubleSort(double *tomb, int len) {  // complexity: n*n, ascending
order
  int idxI, idxJ;

    for(idxI=0; idxI<len; idxI++) {
        for (idxJ=len-1; idxJ>idxI; idxJ--) {
            if (tomb[idxJ-1]>tomb[idxJ])
                 swap(tomb, idxJ-1, idxJ);
        }
    }
}
int main() {
    double myArray[]={22, 16, -7, -42, 6},
           myArray2[]={22, 16, -7, -42, 6};
    int size = sizeof myArray / sizeof (double); // works only with
static arrays

    printf("The original myArray: ");
    print(myArray, size);
    minSort(myArray, size);
    printf("\nThe min sorted myArray: ");
    print(myArray, size);

    printf("\nThe original myArray: ");
    print(myArray2, size);
    boubleSort(myArray2, size);
    printf("\nThe bubble sorted myArray: ");
    print(myArray, size);
    return 0;
}
```

8.1.2.

```
#include <stdio.h>
void combSort(int *, int);
void insertionSort(int *, int);
int main() {
    int idxI;
```

```
    int myArray[] = {5,8,3,4,12,45,87};
    int myArray2[] = {5,8,3,4,12,45,87};
    int size = sizeof(myArray) /sizeof(int);

    combSort(myArray2, size);
    for (idxI = 0; idxI < size; ++idxI)
        printf("%d, ", myArray2[idxI]);
    printf("\n");

    insertionSort(myArray, size);
    for (idxI = 0; idxI < size; ++idxI)
        printf("%d, ", myArray[idxI]);
    printf("\n");
    return 0;
}

void combSort(int * myArray, int size){
    int gap = size, idxI, idxJ, swap, temp;
    for (;;)
    {
        gap = (gap *10) /13;
        if (gap == 9 || gap == 10)
            gap = 11;
        if (gap < 1) gap=1;
        swap = 0;
        for (idxI = 0; idxI < size -gap; idxI++)
        {
            int idxJ = idxI+gap;
            if (myArray[idxI] > myArray[idxJ])
            {
                temp = myArray[idxI];
                myArray[idxI] = myArray[idxJ];
                myArray[idxJ] = temp;
                swap = 1;
            }
        }
        if (gap == 1 && !swap) break;
    }

}
void insertionSort (int * myArray, int size) {
    int idxI, idxJ, leftBorder, rightBorder, tempSize, actualElement;

    for (idxI = 1; idxI < size; idxI++) {
        actualElement = myArray[idxI];
        leftBorder = 0;
        rightBorder = idxI-1;
        while (leftBorder <= rightBorder) {
            tempSize = (leftBorder+rightBorder) / 2;
            if (actualElement < myArray[tempSize])
                rightBorder = tempSize-1;
            else
                leftBorder = tempSize+1;
        }
        //mindig egy elemtől balra szúrok be, ezért a tőle jobbra levő
elemeket eltolom eggyel
        for (idxJ = idxI-1; idxJ >= leftBorder; idxJ--)
            myArray[idxJ+1] = myArray[idxJ];
        myArray[leftBorder] = actualElement;
```

```
    }
}
```

8.2.1.

```
#include <stdio.h>

void print(double* tomb, int size) {
    int idxI;
    printf("[");
    for (idxI=0; idxI<size; idxI++)
        printf("%lf ", tomb[idxI]);
    printf("]");

}

void swap(double* tomb, int id1, int id2) {
    double temp;
    temp=tomb[id1];
    tomb[id1]=tomb[id2];
    tomb[id2]=temp;
}

int partition(double* tomb, int left, int right) { // more clear method:
create a new array and copy values into the begining and ending from the
old one, and copy back the values into the original array
    double val=tomb[left];
    int lm = left-1;  //left margin
    int rm = right+1;
    while (1) {
        do
            rm--;
        while (tomb[rm] > val);

        do
            lm++;
        while(tomb[lm] < val);

        if(lm < rm)
            swap(tomb, rm, lm);
        else
            return rm;
    }
    return 0; // this point is never reached
}

void quickSort(double* tomb, int left, int right) { // left=first valid
index, right=last valid index
    static int level=0;
    int idxI;
    level++;
    for (idxI=0; idxI<level-1; idxI++)
        printf("\t");
    printf("Begin of quickSort at level %d, [%d, %d]\n", level, left,
right);
    if(left < right) {
        int split_pt = partition(tomb,left, right);
        quickSort(tomb, left, split_pt);
        quickSort(tomb, split_pt+1, right);
    }
```

```
    for (idxI=0; idxI<level-1; idxI++)
        printf("\t");
    printf("End of quickSort at level %d, [%d, %d]\n", level, left,
right);
    level--;
}

int main() {
    double myArray[]={0, -6, 7, -42, 6, 9};
    int size = sizeof myArray / sizeof (double); // works only with
static arrays

    printf("The original myArray: ");
    print(myArray, size);
    quickSort(myArray, 0, size-1);
    printf("\nThe quick sorted myArray: ");
    print(myArray, size);
    return 0;
}
```

8.2.2.

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

void fibonacci(long int * fibonacciArray, long int first, long int
second, int count) {
    static int level = 0;
    long int sum;
    int idxI;
    if (count) {
        level++;
        for (idxI = 0; idxI < level - 1; idxI++)
            printf("\t");
        printf("Begin of Fibonacci series at level %d\n", level);
        fibonacciArray[count + idxI + 1] = second + first; //hazugsag, az
utolso szamolas tartalmazza az elozoeket is, igy csak ennek az ertekei
fognak a tombbe irodni
        fibonacci(fibonacciArray, second, first + second, count - 1);
        for (idxI = 0; idxI < level - 1; idxI++)
            printf("\t");
        printf("End of Fibonacci series at level %d\n", level);
        level--;
    }
}

int main() {
    int idxI;
     int maxIndex;
     long int *fibonacciArray;

    printf("Max index of the Fibonacci series? ");
     scanf("%d", &maxIndex);
    if (maxIndex < 3) {
        for (idxI = 0; idxI < maxIndex; ++idxI)
            printf("%d. element: %d\n", idxI+1, 1);
    }
    else {
        fibonacciArray = (long int *) malloc (sizeof(long int) *
maxIndex);
```

```
        fibonacciArray[0] = fibonacciArray[1] = 1;
        for (idxI = 3; idxI <= maxIndex; ++idxI) { //az elemek
meghatarozasa
            printf("\n");
            fibonacci(fibonacciArray, 1, 1, idxI-2); //alapallapot
        }
        printf("\n");
        for (idxI = 0; idxI < maxIndex; ++idxI) //tagok kiiratasa
            printf("%d. element: %ld\n", idxI + 1, fibonacciArray[idxI]);
        free(fibonacciArray); //memoriafelszabaditas
    }
    return 0;
}
```

8.3.1.

```
#include <stdio.h>
#include <string.h>
int main() {
    int n, idxI, idxJ;
    char name[10][30], tempName[30];

    printf("Number of names: ");
    scanf("%d",&n);
    for(idxI = 0; idxI < n; idxI++) {
        printf("%d. name:", idxI+1);
        scanf("%s", name[idxI]);
    }
    for(idxI = 0; idxI < n; idxI++)
        for(idxJ = 0; idxJ < n; idxJ++) {
            if (strcmp(name[idxI], name[idxJ]) < 0) {
                strcpy(tempName, name[idxI]);
                strcpy(name[idxI], name[idxJ]);
                strcpy(name[idxJ], tempName);
            }
    }
    printf("\nThe ordered names are:\n");
    for(idxI = 0; idxI < n; idxI++)
        printf("%s\n", name[idxI]);
    return 0;
}
```

8.3.2.

```
#include <stdio.h>
#include <string.h>
int main() {
    int numberOfNames, idxI, idxJ;
    char name[10][30], prefix[10][30], tempName[30];

    printf("Number of names: ");
    scanf("%d", &numberOfNames);
    for (idxI = 0; idxI < numberOfNames; idxI++) {
        printf("%d. prefix: ", idxI + 1);
        scanf("%s", prefix[idxI]);
        printf("%d. name:", idxI + 1);
        scanf("%s", name[idxI]);
    }
    for (idxI = 0; idxI < numberOfNames; idxI++) {
        for (idxJ = 0; idxJ < numberOfNames; idxJ++) {
            if (strcmp(name[idxI], name[idxJ]) < 0) {
```

```
                strcpy(tempName, name[idxI]);
                strcpy(name[idxI], name[idxJ]);
                strcpy(name[idxJ], tempName);
                strcpy(tempName, prefix[idxI]);
                strcpy(prefix[idxI], prefix[idxJ]);
                strcpy(prefix[idxJ], tempName);
            }
        }
    }
    printf("\nThe ordered names are:\n");
    for (idxI = 0; idxI < numberOfNames; idxI++)
        printf("%s %s\n", prefix[idxI], name[idxI]);
    return 0;
}
```

8.4.1.

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

typedef struct {
    int ID;
    double size;
    char origin[100];
} DataType;

typedef struct le {
    DataType* data;
    struct le* next;
    struct le* prev;
} listElem;

typedef struct {
    struct le head; // sentinel
    struct le tail; // sentinel
} List;

void displayListElem(DataType* data) {
    printf("(ID: %d, ", data->ID);
    printf("size: %lf, ", data->size);
    printf("origin: %s)", data->origin);
}

void deleteList(List* myList) {
    listElem *act;
    act = myList->head.next;
    if (act == &myList->tail) return; // no list to delete
    while (act != &myList->tail) {
        act = act->next;
        free(act->prev);
    }
}

void listList(List* myList) {
    listElem *act=myList->head.next;
    printf("List elments: ");
    while (act != &myList->tail) {
        displayListElem(act->data);
        act = act->next;
```

```
        if (act != &myList->tail)
            printf(", ");
    }
    printf("\n");
}

void insert(List* myList, DataType* newData) {
    listElem *act=myList->head.next, *newElem=NULL, *beforeElem=NULL,
*afterElem=NULL;
    while (act != &myList->tail && act->data->ID > newData->ID) {
    // beware the order of the conditions are important
    // first we have to check if act is a valid element, only then can we
check it's ID field
        act = act->next;
    }
    afterElem = act;
    beforeElem = act->prev;
    newElem = (listElem*)malloc(sizeof(listElem));
    newElem->data = newData;
    beforeElem->next = newElem;
    newElem->next = afterElem;
    afterElem->prev = newElem;
    newElem->prev = beforeElem;
}

int main() {
//  List myList={{NULL, NULL, NULL}, {NULL, NULL, NULL}};
    List myList={NULL}; // do the same as the previous row
    myList.head.next = &myList.tail;
    myList.tail.prev = &myList.head;
    DataType testData[]={{1, 2.3, "Hu"}, {7, 12.3, "Ro"}, {0, 45.3,
"Eu"}, {5, 14.1, "Gb"} };
    int idxI;

    for (idxI=0; idxI<4; idxI++) {
        printf("Insert ");
        displayListElem(&testData[idxI]);
        printf("\n");
        insert(&myList, &testData[idxI]);
        listList(&myList);
        printf("\n");
    }
    deleteList(&myList);
    return 0;
}
```

8.4.2.

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

typedef struct {
    int ID;
    double size;
    char origin[100];
} DataType;

typedef struct le {
    DataType* data;
```

```
    struct le* next;
    struct le* prev;
} listElem;

typedef struct {
    struct le head; // sentinel
    struct le tail; // sentinel
    int count;
} List;

void displayListElem(DataType* data) {
    printf("(ID: %d, ", data->ID);
    printf("size: %lf, ", data->size);
    printf("origin: %s)", data->origin);
}

void deleteList(List* myList) {
    listElem *act;
    act = myList->head.next;
    if (act == &myList->tail) return; // no list to delete
    while (act != &myList->tail) {
        act = act->next;
        free(act->prev);
    }
}

void listList(List* myList) {
    listElem *act=myList->head.next;
    printf("List elments: ");
    while (act != &myList->tail) {
        displayListElem(act->data);
        act = act->next;
        if (act != &myList->tail)
            printf(", ");
    }
    printf("\n");
    printf("Element count: %d\n", myList->count);
}

void insert(List* myList, DataType* newData) {
    listElem *act=myList->head.next, *newElem=NULL, *beforeElem=NULL,
*afterElem=NULL;
    while (act != &myList->tail && act->data->ID > newData->ID) {
    // beware the order of the conditions are important
    // first we have to check if act is a valid element, only then can we
check it's ID field
        act = act->next;
    }
    afterElem = act;
    beforeElem = act->prev;
    newElem = (listElem*)malloc(sizeof(listElem));
    newElem->data = newData;
    beforeElem->next = newElem;
    newElem->next = afterElem;
    afterElem->prev = newElem;
    newElem->prev = beforeElem;
}

int main() {
```

```
//  List myList={{NULL, NULL, NULL}, {NULL, NULL, NULL}};
    List myList={NULL}; // do the same as the previous row
    myList.count = 0;
    myList.head.next = &myList.tail;
    myList.tail.prev = &myList.head;
    DataType testData[]={{1, 2.3, "Hu"}, {7, 12.3, „Ro”}, {0, 45.3,
"Eu"}, {5, 14.1, "Gb"} };
    int idxI;

    for (idxI=0; idxI<4; idxI++) {
        printf("Insert ");
        displayListElem(&testData[idxI]);
        printf("\n");
        insert(&myList, &testData[idxI]);
        myList.count += 1;
        listList(&myList);
        printf("\n");
    }
    printf("Count: %d\n", myList.count);
    deleteList(&myList);
    return 0;
}
```

8.5.1.

```
#include <stdio.h>
#include <malloc.h>
typedef struct {
    int ID;
    char name[50];
    float price;
} headphone;

void exchange(headphone** a, int i, int j) {
    headphone *temp=a[i]; // no structure copy
    a[i]=a[j];
    a[j]=temp;
}

void downheap(headphone** a, int size, int parent) {
    int left=2*parent+1, right=2*parent+2, maxChild=left;
    if (left>=size) return; // a[v] has no children
    // if there are one child then left is the maxChild
    if (right<size && a[right]->ID > a[left]->ID) maxChild=right;
    // if there are two children and the right is bigger, then it will be
maxChild
    if (a[parent]->ID >= a[maxChild]->ID) return; // it is a heap
    else {
        exchange(a, parent, maxChild); // force heap property at the top
        downheap(a, size, maxChild); // force heap property in the changed
subtree
    }
}

void buildheap(headphone** a, int n) {
    int v;
    for (v=n/2-1; v>=0; v--)
        downheap(a, n, v);
}
```

```
void heapsort(headphone** a, int n) {
    buildheap(a, n);
    while (n>1) {
        n--;
        exchange(a, 0, n);
        // put the biggest of the current heap into its final place in the
array
        // put the most right leaf at the top of the heap
        downheap(a, n, 0);
        // force the heap property on the decrease heap
    }
}

void printHeadphone(headphone* node) {
    if (!node) return;
    printf("[ID: %3d, ", node->ID);
    printf("name: %10s, ", node->name);
    printf("price: %6.2f]", node->price);
}

void printArray(headphone** array, int size) {
    int idxI=0;
    for (;idxI<size;idxI++) {
        printHeadphone(array[idxI]);
        printf("\n");
    }
}

int main() {
    headphone b={23, "Genius", 2710}, c={7, "MS", 3250},
    d={63, "Verano", 1160}, e={11, "LG", 6980}, f={9, "Samsung", 2370},
    g={42, "Ele", 1260}, h={12, "Azona", 1230};
    headphone* a[]={&b, &c, &d, &e, &f, &g, &h};
    int size = sizeof (a) / sizeof(headphone*);

    printf("Unordered heap:\n");
    printArray(a, size);
    heapsort(a, size);
    printf("\nOrdered heap:\n");
    printArray(a, size);
    return 0;
}
```

8.6.1.

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

#define DEFAULT_INPUTFILE "words.txt"
#define MAX_LENGTH 16
#define TRUE 1
#define FALSE 0

typedef char TWord[MAX_LENGTH];

// A > B
int Greater(char * A, char * B)
{
  int i;
```

```
  int sa = strlen(A);
  int sb = strlen(B);
  int min = sa < sb ? sa : sb;
  for (i = 0; i < min; i++)
  {
    if (A[i] > B[i])
      return TRUE;
    if (A[i] < B[i])
      return FALSE;
  }
  return sa > sb;
}

int ReadWords(FILE * fd, TWord ** W)
{
  int num, i;
  fscanf(fd, "%d", &num);
  (*W) = (TWord *)malloc(sizeof(TWord) * num);
  for (i = 0; i < num; i++)
    fscanf(fd, "%s", (*W)[i]);
  return num;
}

void PrintWords(TWord * W, int num)
{
  int i;
  for (i = 0; i < num; i++)
    printf("\t%d.: %s\n", i + 1, W[i]);
}

void Sort(TWord * W, int num)
{
  int min, i, j;
  TWord Temp;
  for (i = 0; i < num - 1; i++)
  {
    min = i;
    for (j = i; j < num; j++)
    {
      if (Greater(W[min], W[j]))
        min = j;
    }
    strcpy(Temp, W[min]);
    strcpy(W[min], W[i]);
    strcpy(W[i], Temp);
  }
}

int main(int argc, char * argv[])
{
  TWord * Words;
  int WNum;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
    perror(„Error”);
    return 0;
  }
  WNum = ReadWords(fd, &Words);
```

```
  fclose(fd);
  printf("Read words:\n");
  PrintWords(Words, WNum);
  Sort(Words, WNum);
  printf("\n***************\nSorted words:\n");
  PrintWords(Words, WNum);
  free(Words);
  Words = NULL;
  return 0;
}

// words.txt
7
WARIOR
APPLE
SZOKOZSHIP
BOOK
SZOKOZ
PHISIC
LIMON
```

8.7.1.

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

#define DEFAULT_INPUTFILE "memo1.txt"
#define MAX_MEMO_LEN 21
#define TRUE 1
#define FALSE (!TRUE)

struct TMemo
{
  int Year;
  int Month;
  int Day;
  int Hour;
  char m[MAX_MEMO_LEN];
};

int ReadMemos(FILE * fd, struct TMemo ** M)
{
  int c, i;
  struct TMemo * mptr;
  fscanf(fd, "%d", &c);
  (*M) = mptr = (struct TMemo*)malloc(sizeof(struct TMemo) * c);
  for (i = 0; i < c; i++)
  {
    fscanf(fd,"%d %d %d %d", &mptr->Year, &mptr->Month, &mptr->Day,
&mptr->Hour);
    fscanf(fd, "%s", mptr->m);
    mptr++;
  }
  return c;
}

void PrintMemos(struct TMemo * M, int c)
{
  int i;
```

```
  for (i = 0; i < c; i++)
  {
    printf("Year: %d\nMonth: %d\nDay: %d\nHour: %d\n\t %s\n\n", M->Year,
M->Month, M->Day, M->Hour, M->m);
    M++;
  }
}

// A > B
int Greater(struct TMemo * A, struct TMemo * B)
{
  if (A->Year != B->Year)
  {
    return (A->Year > B->Year);
  } else
  {
    if (A->Month != B->Month)
    {
      return (A->Month > B->Month);
    } else
    {
      if (A->Day != B->Day)
      {
        return (A->Day > B->Day);
      } else
      {
        if (A->Hour > B->Hour)
          return (A->Hour > B->Hour);
      }
    }
  }
  return FALSE;
}

void Sort(struct TMemo * M, int c)
{
  int i, j, mini;
  struct TMemo tmp;
  for (i = 0; i < c - 1; i++)
  {
    mini = i;
    for (j = i; j < c; j++)
    {
      if (Greater( M + mini, M + j))
        mini = j;
    }
    memcpy(&tmp, M + mini, sizeof(struct TMemo));
    memcpy(M + mini, M + i, sizeof(struct TMemo));
    memcpy(M + i, &tmp, sizeof(struct TMemo));
  }
}

int main(int argc, char * argv[])
{
  struct TMemo * Memos;
  int MCount = 0;
  FILE * fd = fopen(argc > 1 ? argv[1] : DEFAULT_INPUTFILE, "r");
  if (fd == NULL)
  {
```

```
    perror("Error");
    return 0;
  }
  MCount = ReadMemos(fd, &Memos);
  fclose(fd);
  Sort(Memos, MCount);
  PrintMemos(Memos, MCount);
  free(Memos);
  Memos = NULL;
  return 0;
}
```