Adatszerkezetek és algoritmusok példatár

Verzió: 0.07.f - 2014.december 14.

Verzió magyarázat: N.NN.X

- *N: főverzió, ez ebben a félévben 0 lesz.*
- *NN: a gyakorlatok (hetek) előrehaladtával növekszik egyesével*
- *X módosítások jelölésére, verziónként visszaugrik „a”-ra.*

Tartalom

Megjegyzés - Előszó

Ez a példatár tartalmazza a gyakorlatok során bemutatott nagyobb feladatokat és házi feladatokat, amelyekkel az új anyag könnyebben elsajátítható.

Ezen túl tartalmaz elméleti és gyakorlati kérdéseket, amelyek korábban papír-toll, illetve kisZH kérdések voltak. (A zh-n ezek a kérdések ***is*** előfordulhatnak, valamint ehhez hasonló és hasonló stílusú kérdésekkel lehet találkozni.)

A harmadik fő rész, pedig olyan feladatokat gyűjt össze, amelyek a géptermi zh-n fordultak elő korábban. Ilyen nehézségű, illetve típusú feladatok várhatók.

Felhívjuk mindenki figyelmét, hogy ennek a példatárnak a megoldása nem garantálja a zh sikerességét, azonban az esélyeket jelentően növeli. Fontos az önálló problémamegoldó készség, az algoritmizáló készség és a kitartás egyaránt. Továbbá a zh feladatok ***nem garantáltan*** ezekből a feladatokból kerülnek kiválasztásra.

Sikeres felkészülést kívánnak a szerkesztők és gyakorlatvezetők!

Gyakorlati feladatok

Első gyakorlat

01. Feladat (G01F01)

Írjunk egy példaprogramot, amely

- Bekér egy számot a felhasználótól és ennek megfelelően létrehoz egy tömböt
- A tömbbe feltölti azokat a számokat, amelyeket a felhasználó megad
- Ezt követően a tömböt paraméterként át kell adni egy függvénynek, amely
  - Az átvett tömb értékeit lemásolja egy új tömbbe
  - A másolatba minden érték kétszerese kerüljön bele, kivéve, ha a kétszeres érték osztható 10-tel
  - A függvény gondoskodjon arról, hogy ne hagyjon maga után memóriaszemetet
- A visszakapott új tömb egy másik függvény inputja legyen, amely
  - Meghatározza a tömbben található számok átlagát
  - Az átlagértékkel térjen vissza a függvény
- A program ne hagyjon maga után memóriaszemetet

02. Feladat (G01F02)

Írj egy olyan programot, amely egy mátrixot tárol

- A mátrix méretét és az elemeket előre nem ismerjük
- Az adatokat egy „in.txt” fájlból kell beolvasni. A fájlban az első sorban a sorok száma (n) van, a második sorban az oszlopok száma (m), többi sorban (n×m) pedig a beolvasandó értékek, sorfolytonosan.
- A mátrixot egy tömbben tároljuk el oszlop folytonosan
  - Figyelem! Megváltozik a sorrend, az indexekre gondolni kell
- Írd meg az összeadás és szorzás műveleteket!
- Írj egy transzponálást végrehajtó függvényt!
- Írj egy kiíró műveletet, ami az elemeket áttekinthető formában megjeleníti!
- A főprogramon belül készíts menüt!
  - Egy ciklusban kérdezd meg a felhasználótól, hogy akar-e még műveletet végezni!
  - Ha igen, kérdezd meg, hogy hogy milyen műveletet!
  - Kérj be két mátrixot elemenként, majd végezd el a kért műveletet!
  - Az eredményt jelenítsd meg a képernyőn a kiíró műveletével!

03. Feladat (G01F03)

Hozz létre egy n hosszú tömböt – dinamikus memóriafoglalással

- Töltsd bele az első n prímszámot
- Másold le a tömböt (tényleges másolat)

Emeld négyzetre a másolat elemeit, kivéve, ha a szám háromjegyű

- Ha háromjegyű, akkor legyen a tömbben az új érték nulla
- A másolatot ki kell írni a képernyőre.

Ezt követően írd ki a Pascal háromszög n-edik sorát

A program a futás során kezelje a memóriát helyesen!

04. Feladat (G01F04)

Készíts egy olyan programot, amely

- Négyzetek és téglalapok adatainak tárolására alkalmas
  - Ehhez két tömböt kell használni, ahol a négyszögek két oldalának hosszát tároljuk el
- A két tömb méretét a program indulásakor a felhasználó adja meg, amelyet követően azokat dinamikusan kell létrehozni
- A tömbök értékekkel történő feltöltése után meg kell keresni:
  - A legnagyobb területű téglalapot
  - A legkisebb kerületű négyszöget
  - A megtalált síkidomok oldalainak hosszát és méretét ki kell írni
  - A két keresésre két függvényt kell írni, amelyek paraméterben kapják meg a tömböket
- Ügyelni kell arra, hogy ne legyen memóriaszivárgás és ne legyen felesleges memóriahasználat
- Legyen még egy függvény, amely paraméterként átveszi a tömböt és meghatározza a területértékek átlagát
  - Ezt ki is kell írni

05. Feladat (G01F05)

Készíts egy programot, ami

- Képes tárolni egy összefüggő irányítatlan gráfot
- A gráf leírását tömbök tömbjével kell megoldani, dinamikus memóriakezeléssel
  - Az i-edik tömb j-edik értéke 1 vagy 0.
  - Ha 1, akkor az i és j pont között van él, ha 0, akkor nincsen
  - Ha az i-edik tömb j-edik értéke 1, akkor a j-edik tömb i-edik értéke is 1.
- A program indításkor kérdezze meg, hogy mennyi csúcsot szeretne tárolni a felhasználó
  - Ennek megfelelően hozza létre a tömböket és töltse fel 0 értékekkel
- Ezt követően legyen lehetőség élek felvételére
  - Természetesen ügyeljen a program, hogy érvényesek legyenek a bevitt értékek
- A program legyen képes eldönteni, hogy két tetszőleges csomópont között húzódik-e pontosan kettő hosszú út!

Második Gyakorlat

06. Feladat (G02F01)

Az eddig tanultak alapján készítsük el az ábrán látható osztályokat a szokásos műveletekkel!

- Az AbstractArray–mint ahogy neve is sugallja –legyen absztrakt.
- FixedArray: ne legyen megváltoztatható a kezdeti mérete.
- DynamicArray: ha a kezdeti lefoglalt memória nem elég, foglaljon le 2x annyit.

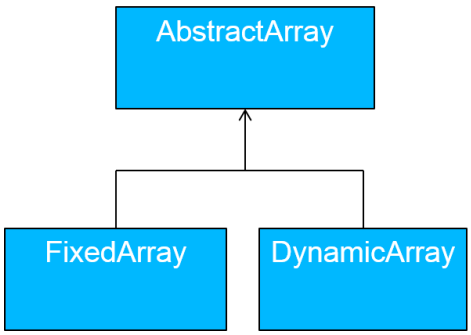
07. Feladat (G02F02)

- Hozzatok létre egy absztrakt „Alakzat” osztályt, ennek legyen 2 metódusa, a kerulet_szamol() és a terulet_szamol()
- Hozzatok létre egy „Haromszog”, egy„Teglalap”, és egy „Kor” osztályt, amik az alakzat osztályból származnak.
- A „Teglalap” osztályból származzon le egy „Negyzet” osztály is.
- A kört leszámítva tartalmazzák az osztályok az oldalaik hosszát, a kör pedig egy sugarat.
- A main.cpp-ben hozzatok létre egy „Alakzat” pointer, vagy referencia tömböt, amit töltsetek fel tetszőleges alakzatokkal, és keressétek meg benne a legkisebb területűt, és a legnagyobb kerületűt!
- Írjatok olyan operátort minden alakzathoz, ami kiírja az alakzat nevét, és paramétereit a „cout”-ra! Erre is csináljatok néhány próbát!

08. Feladat (G02F03)

Írjunk egy iskolát modellező alkalmazást!

- Először is az iskolába járnak emberek, akiknek van nevük, születési évük és nemük.
- Az iskolában tanítanak tantárgyakat, amelyeknek van nevük, leírásuk és hozzá vannak rendelve évfolyamok, amikor ezeket oktatni kell.
- Az iskolába járó emberek között vannak tanulók, akik egy osztályba tartoznak, akiknek van egy tanulmányi átlaguk, és akiknél el kell tárolni a szülő/nevelő elérhetőségeit.



- Az iskolában tanítanak tanárok, akikhez bizonyos tárgyak vannak hozzárendelve. A tanárok között vannak osztályfőnökök, akiknek van osztályuk. A tanároknak van heti óraszámuk, órabérük, ami alapján a havi fizetésük kiszámolható.
- A tanárok között van egy igazgató, akinek a fizetése nem csak az órabér és a heti óraszám alapján képződik (van valamifajta plusz fizetés).
- Végül van az iskola, amiben vannak tanulók, tanárok, igazgató, aminek van egy címe, neve és ahol kiszámolható, hogy havonta összesen mennyi fizetést kell kiosztani.

09. Feladat (G02F04)

Tervezzünk meg és írjunk meg egy alkalmazott osztályt!

Útmutatás:

- Gondold át, hogy egy alkalmazottat egy tetszőleges vállalatnál milyen adatokkal lehet reprezentálni.
- Ha ez megvan, gondold át azt, hogy mik lehetnek azok az eljárások/műveletek, amiket ehhez az alkalmazott osztályhoz hozzá lehetne kapcsolni. Gondold át, hogy a meglévő adattagokon milyen értelmes műveleteket/lekérdezéseket lehetne végrehajtani.

- Gondold át, hogy a fentiek összességéből melyek azok, amelyek minden alkalmazott esetén ugyanazok/ugyanazt az eredményt/hatást érik el, és melyek azok, amelyek minden alkalmazott esetén különbözőek.
- Majd gondold át, hogy mi az, ami hozzátartozna az osztály publikus interfészéhez, és mi az, amit érdemes elrejteni.
- Ha mindez megvan, a megfelelő nyelvi elemeket felhasználva írd meg az alkalmazott osztály!

Feladat: A megírt Alkalmazott osztályból leszármaztatva valósítsunk meg egy Főnök osztályt!

Útmutatás:

- Melyek azok a változók, amelyeket célszerű lenne elfedni?
- Milyen új változókat lehetne bevezetni?
- Melyek azok a metódusok, melyeket felül kéne definiálni?
- Milyen új metódusokat kellene megírni?

Majd ezután írj egy olyan main függvényt, melyben kipróbálod a polimorfizmus és a dinamikus kötés lehetőségeit!

10. Feladat (G02F05)

Készítsünk egy mátrix osztályt:

- a konstruktor paramétereivel megadhatjuk méreteit, amiket aztán le is lehet kérdezni,
- legyen assignment operátora és copy constructora,
- legyen ==, + operátora,
- a () operátorral lehessen elérni az értékeit (amit aztán be is lehet állítani),
- legyen << operátora, amivel streamre írhatunk,
- végül legyen az osztálynak egy statikus függvénye, amivel egységmátrixokat lehet gyártani.

11. Feladat (G02F06)

Három féle személy típusunk van (diák, dolgozó és nyugdíjas) és három féle járművünk (taxi, busz, vonat). Hogy az összetűzéseket elkerüljük a diákok csak vonat, a dolgozók csak taxival a nyugdíjasok pedig csak busszal tudnak utazni.

Jármű	Átlagsebesség [km/h]	Díj / km [Ft]	Férőhely [fő]
Taxi	60	200	3
Busz	45	55	8
Vonat	90	35	15

Feladat: a program generál i db diákot, j db dolgozót és k db nyugdíjast.

- i, j és k értékek tudatában, annyi taxit, buszt és vonatot kell létrehozni, amennyi el tudja szállítani az utasokat. A járművek létrehozásakor, kapnak egy 0,5-1,5 közötti random értéket. Ezzel az értékkel megszorozzuk az átlagsebességüket így különböző sebességgel fognak haladni.

A fuvarozó társaságok számára fontos, hogy az utasok elégedettek legyenek.

- Egy utas akkor elégedett, ha a megadott távolságra az átlagos utazási idő alatt eljut. (idő = távolság / átlagsebesség). Ha ezt az időt 15 % -kal meghaladjuk, akkor az utasok elégedetlenek lesznek.
- Taxi esetén az egy főre jutó útiköltség függ az utasok számától, így az elégedettség mérésénél ezt is figyelembe kell venni.

Tehát az utasok beszállnak a járművekbe, elfuvarozzák őket adott távolságra (ezt a jármű fuvar függvénye kapja paraméterül), majd a fuvar végén az utasok elmondják, hogy elégedettek e.

A könnyebbség kedvéért egy járműben minden utas ugyanaddig utazik.

Harmadik Gyakorlat

12. Feladat (GF0301)

Dinamikus verem:

- Nyissátok meg a projektet, és implementáljátok (új osztályként, új fájlba) a verem statikus megvalósítását úgy, hogy a dynamic_stack_main.cpp helyesen lefusson!
- Figyeljetek azokra az esetekre, ahol a feladatot nem lehet (specifikáció szerint) végrehajtani! (kivételkezelés)

13. Feladat (GF0302)

Statikus sor:

- Nyissátok meg a projektet, és (új osztályként, új fájlba) implementáljátok, írjátok meg a sor statikus megvalósítását úgy, hogy a *static_queue_main.cpp* helyesen lefusson!
- Figyeljetek azokra az esetekre, ahol a feladatot nem lehet (specifikáció szerint) végrehajtani! (kivételkezelés)

14. Feladat (GF0303)

Lengyelforma gyakorlása:

- Írjátok át a VEREM és a SOR dinamikus implementációját is úgy, hogy azok char típusú elemeket tudjanak tárolni
- Ezeket felhasználva írjátok meg az egyszerűsített lengyelformára hozó algoritmust
    - Tipp: Egy jegyű számokkal dolgozzatok!

15. Feladat (GF0304)

- Adott egy fájl amiben titkosírással elrejtettünk egy üzenetet. Verem és sor használatával, fejtsd vissza mit titkosítottunk.
- A fájlban #, & és @ szimbólumok valamint a következő karakterek szerepelhetnek: a-z, A-Z, 0-9
- Minden karakter két szimbólum közé esik.
- A megfejtés azon karakterek (a fájlban szereplésüknek megfelelő sorrendben) összeolvasása amelyeket két azonos szimbólum határol.
- Példa a fájlból:
    - #a&#I@#Q@&A##v@@7#@D&#y&@k&@8##W@&V@
    - &i@#H#@V&@q&@s#@D#@q&@F##s&&v##e@#L&#a&
    - &7##p&&J@#i&&n@@R##M@@f#&j@&V##Y&&T@&4##l
- Az eredményt írd ki a konzolra és egy fájlba.

16. Feladat (GF0305)

- Készítsd el a deque implementációját a statikus sor módosításával. A deque kétirányú sor, tehát mindkét végére tehetünk be és vehetünk ki onnan elemet, de a középső elemeket nem érjük el.
- A deque működésének bemutatásához készíts egy piros-zöld papucs kártyajátékot szimuláló programot. (Ez a piros papucs nevű kártyajáték kicsit módosított változata.)
- A játékot két játékos játssza egy pakli (32 lapos) magyar kártyával. A kártyalapok értéke nem, csak a színe (piros, zöld, tök, makk) számít.
- A lapokat először szétosztják maguk között két megkevert, egyenlő méretű pakliba (figyelj rá, hogy minden színből 8 db legyen). Ezután felváltva pakolnak kártyát a saját paklijuk **tetejéről** az asztalon levő kupac **tetejére**.
- Ha valaki *pirosat* rakott, akkor a másik felveszi az asztalon levő teljes paklit, és sorban elhelyezi a saját paklija **aljára**, **felülről lefelé**. Ha valaki *zöldet* rakott, akkor szintén a másik veszi fel a teljes paklit, de fordított sorrendben (**alulról felfelé**) helyezi el a saját paklija **aljára**.
- A játéknak vége, ha elfogyott valamelyik játékos paklija, és ő következik soron; ekkor ő nyert (azaz lehet „visszahívni” 1 körig). Ha 200 lépés után sem fogyott el senkinek a kártyája, akkor a játékosok megunják, és kiegyeznek egy döntetlenben.
- A megvalósításban a játékosok kezében levő paklikat egy-egy sorral, az asztalon levő paklit pedig egy kétirányú sorral valósítsd meg! A kártyák számértékét nem szükséges figyelembe venned.

Negyedik gyakorlat

17. Feladat (GF0401)

Pályaudvar szimulátor:

- A pályaudvar fix darabszámú vágányból áll
- Egy vágányon egyszerre több vonat is tartózkodhat
- Egy vágányról az a vonat indulhat el először (értelemszerűen), aki elsőnek érkezett
- A pályaudvar vágányaira vonatok érkeznek, amelyek különböző darabszámú vagonnal rendelkeznek
- Egy vagon tartalma szöveges. pl.: 'biciklis kocsi', 'marhavagon', 'első osztály', stb. Nem kell külön osztály a vagonoknak!
- A vonathoz bárhova hozzá lehet illeszteni egy új vagont. Kiválasztjuk az aktuális vagont, és elé, vagy utána új vagont illesztünk be
- A program véletlenszerűen érkeztet néhány vonatot a pályaudvarra (Egy vágányra több vonat is érkezzen!)
- Néhány vágányon egy-egy vonathoz tegyen a program egy új vagont! (pl. az első vágányon levő második vonat végéhez hozzáteszi a 'gyerekkocsi' vagont.)
- Néhány véletlenszerű vágányról elindít egy-egy vonatot!

18. Feladat (GF0402)

Egyszerű adatbázis

Készíts egy adatbázis programot, ami egy eltárolt fájlt képes beolvasni, a benne levő elemeket módosítani, majd visszaírni a fájlba.

Az adatbázisban tárolt elemek egyszerű string-ek.

A fájlban az elemeket soronként tárold.

A program induláskor beolvassa a fájl tartalmát egy listába.

Ezután egy konzolos menü interfészt ad, amelyen keresztül a felhasználó kiírathatja az összes elemet, eltávolíthat, hozzáadhat.

A program kilépéskor írja be az elemeket a fájlba, így következő induláskor megint láthatóak a tárolt elemek.

19. Feladat (GF0403)

Lista statikus implementációval

- A lista elemeit nem dinamikus láncolással, hanem statikusan, egy tömbben tárold.
- A tömb elemei érték-index párok.
- A lista az összes lista műveletet implementálja.
- Iterátorokkal is rendelkezzen.
- Copy constructor és assignment operator is legyen rajta definiálva.
- Figyelj arra, hogy a statikus ábrázolás miatt a lista nem csak alul-, hanem túl is csordulhat!

20. Feladat (GF0404)

Írj egy befűző függvényt, amely paraméterül kap egy másik listát,és az elemeit befűzi az aktuális elem utánra (és írd meg azt is ami az aktuális elem elé szúrja be)!

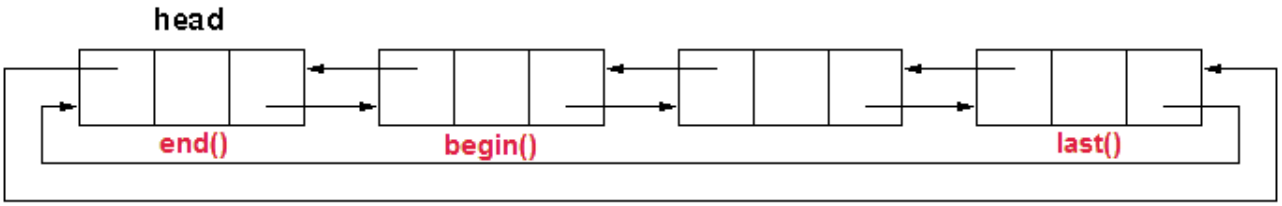
- A függvény legyen a lista tagfüggvénye.
- Profiknak: A függvény ne legyen része az alap listának! Ezen kívül a befűzés konstans időben fusson le, és a befűzendő listát ürítse ki!

Ezek után használd a listát mint stringet, tehát tárolj benne karaktereket (char)!
Kérj be a konzolból egy szöveget, és töltsd fel vele a listát!
Ezután kérj be egy másik szöveget, és egy karaktert, majd a szöveget fűzd be az első szövegbe úgy, hogy az a bekért karakter első előfordulása után kezdődjön!

21. Feladat (GF0405)

Valósítsunk meg kétszeresen, körkörösen láncolt listát fejelemmel!

A körkörös láncolás azt jelenti, hogy a begin() iterátor a fejelem rákövetkezőjére mutat, az end() a fejelemre, a last() pedig a fejelem megelőzőjére



Ez azt eredményezi, hogy a lista végén nincs nullpointer, hanem a láncot „visszaakasztjuk” a lista elejére.

Az insert művelet így lényegesen leegyszerűsödik, hiszen nem kell vizsgálni a külön eseteket, minden elemnek van előző és következő eleme.

Ötödik gyakorlat

22. Feladat (GF0501)

Az alábbi feladat megoldása bináris keresőfával.

- A könyvtárban a könyvekről a következő adatokat kell tárolnunk: (egyedi azonosító), string cím, int hossz, bool ki van-e épp kölcsönözve?
- Implementáld a fa iterátort (inorder járja be a fát) a hossz és cím mezőkre.
- Tölts fel 8 tetszőleges könyvet a könyvtáradba.
- A program futtatása során a következőkre legyen lehetőség:
    - új könyv hozzáadása
    - könyv kikölcsönzése/visszahozása
    - könyvek listázása cím szerint
    - könyv törlése
    - Extra: hossz alapján is lehessen listázni!

23. Feladat (GF0502)

- Hozz létre egy két template paraméteres bináris keresőfát. A két template paraméter itt azt jelenti, hogy az egyik paraméter a kulcs típusa (K), a másik, az ott tárolt érték típusa (V). Tehát ezek után a fa alkalmas kulcs-érték párok tárolására.
- Írj egy olyan alkalmazást, amelyben a felhasználó egy menün keresztül használni tud egy angol-magyar szótárt.
- Az angol-magyar szótárat a következőképpen hozd létre:
- Töltsd fel a megírt két template paraméteres fát úgy, hogy a kulcs az angol szó (string), az érték pedig az angol szó magyar megfelelő(i) (string / list<string>)
- Ezután pedig írj egy menüt, amin keresztül egy felhasználó használni tudja a szótárt. A menünek a következő funkciói legyenek:
    - Ki lehessen keresni kívánt angol szó magyar megfelelő(i)t. Ha ez eddig még nincs benne a szótárban, akkor legyen lehetőség ezen új szó bevitelére.
    - Ki lehessen listázni a szótárban tárolt angol szavakat és ezek magyar megfelelő(i)t, abc sorrendben.
- Ha rákeresünk egy csúcsra a megadott angol szó segítségével, akkor hogyan nyerjük ki a csúcsban tárolt magyar megfelelőjét? Ehhez a következőt kell tenned: definiálj a <K, V> típus paraméteres bináris keresőfa műveletei között egy V getValue(K k) függvényt. (Tehát kulcs alapján keresek, és az értéket nyerem ki.)

24. Feladat (GF0503)

- Hozz létre egy olyan fát, melyben a kulcs string típusú, továbbá minden csomópont tárol egy vector-t, melynek elemei egész számokból álló párok (tehát egy két paraméterű fára lesz szükséged: K kulcs, V érték tárolódik minden csúcsban).
- Írj egy olyan alkalmazást, amely beolvas egy szöveges fájlt. A szövegben található (szóközzel vagy sortöréssel elválasztott) szavakat beilleszti egy bináris kereső fába, és minden szóhoz feljegyzi, hogy melyik sor hányadik karakterénél találta meg a szót (erre használd a számpárok vektorát: sor-pozíció).
- Legyen lehetőség egy adott szó előfordulásainak megkeresésére.
- Legyen lehetőség az összes, szövegben előfordult, különböző szó számának meghatározására.
- Legyen lehetőség annak meghatározására, hogy melyik szó fordult elő a legtöbbször.
- A bemeneti szövegben nem kell készülnöd írásjelekre, csak az angol abc betűi és a szavakat elválasztó szóközök/sortörések lesznek a fájlban.
- Ha rákeresünk egy adott szó előfordulásait tároló csúcsra, akkor hogyan nyerjük ki a csúcsban tárolt vektort? Definiálj a bináris keresőfa műveletei között egy V getValue(K k) függvényt. (Tehát kulcs alapján keresek, és az értéket nyerem ki.)

25. Feladat (GF0504)

- Adott két bináris keresőfa. Az egyikről annyit tudunk, hogy páratlan, a másikról pedig, hogy páros számokat tartalmaz.
- Készíts iterátort a bináris keresőfához!
- Bináris keresőfával és iterátor felhasználásával oldd meg a következőket:
    - A két bemeneti fából készíts egyetlen fát, majd inorder írd ki az eredményt.
    - Az iterátor és a bináris keresőfák felhasználásával készíts egy olyan programot, amely bementre egy számsort (listát) vár, a kimeneten pedig növekvő sorrendben adja azt vissza.

26. Feladat (GF0505)

- Egészítsd ki a Binkerfa implementációt iterátor osztállyal.
- Kivételkezelést nem feledve készítsd el a következő műveleteket:
    - Mozgatás a szülőelemre, jobbgyerekre, balgyerekre
    - Maximumkeresés
    - Tetszőleges elem keresése
    - Iterátor állítása a következő ill. a megelőző elemre (ha létezik)
- Készítsd el a << operátort, ami a fa mindhárom bejárását kiírja.

Hatodik gyakorlat

27. Feladat (G06F01)

Alakítsátok át úgy az AVL fát, hogy kulcs-érték párokat lehessen benne tárolni. Második template paraméterként várja az érték típusát. A kulcshoz tartozó értéket a node-ban tárolja.

Példa a használatára:

```
avl_tree<int, string> szamok;
szamok.insert(6, “hat”);
avl_tree<int, string>::iterator it = szamok.find_it(6);
cout << “6 = “ << it.value();
```

Az így megírt map segítségével

- hozzatok létre egy telefonkönyvet
- szúrjatok bele pár név/telefonszám párost
- keressetek rá egy-egy név telefonszámára
- majd pedig iteráljatok végig a telefonkönyvön és írjátok ki az elemeit a konzolra.

28. Feladat (G06F02)

AVL fákat, mint halmazokat felhasználva valósítsd meg az unió, metszet és különbség műveleteket!

Ehhez tölts fel két AVL fát véletlen számokkal, majd végezd el a műveleteket!

Művelet: Két AVL fa paraméter, és egy AVL fa visszatérési érték! (Vagyis valóban művelet!)

(megj.: Mivel két egyforma csúcs nem lehet, ezért mondhatjuk, hogy halmazok!)

29. Feladat (G06F03)

AVL fák segítségével készíts egy angol-magyar illetve egy magyar-angol szótár adatbázist.

A fordítandó szavai legyenek a csomópontok kulcsai és a hozzájuk tartozó fordítás pedig az értékek.

- Legyen lehetőség arra, hogy akár több hasonló jelentésű fordítást is felvegyünk egy adott szóhoz.

Készíts felhasználóbarát menüt, ami alkalmas a szótárak használatára:

- Tudjunk keresni egy adott szót és kiírni a hozzá tartozó fordításokat.
- Ha a keresett szó nem szerepel a fában, akkor vegyük fel legalább egy fordítással.

30. Feladat (G06F04)

Készíts valamilyen iterátort a fához! Egyéni választásod szerint lehet külön iterátor osztály, amelyből több példány is lehet, vagy egyetlen act mutató az AVL fa tagjaként, ahogyan a láncolt listánál láttuk.

Minimális műveletkészlet az iterátornak:

- Fa legkisebb elemére állítás
- Rákövetkező elemre állítás
- Mutatott csúcs értékének lekérdezése
- Lekérdezni, hogy túlfutottunk-e az utolsó elemen

31. Feladat (G06F05)

Generáljatok kellően sok véletlen adatot, majd szúrjátok be egy AVL fába. (Figyeljünk arra, hogy az AVL fa kiszűri az ismétlődéseket! Ha véletlenül egy már létező elemet hoztunk létre, akkor generáljunk helyette újat!)

Készítsetek statisztikát, hogy mennyi időbe telt egy elem átlagos beszúrása, összesen mennyi idő kellett az AVL fa felépítéséhez, hány forgatásra volt szükség a kiegyensúlyozásokhoz.

Ha elkészült az AVL fa akkor olvassátok ki az elemeket sorrendben egy vektorba.

A most már rendezett elemeket ismét szúrjátok be egy új AVL fába és készítsétek el ugyan azt a statisztikát, mint első esetben.

Értékeljétek ki (gondolatban) a tapasztaltakat!

Hetedik gyakorlat

32. Feladat (G07F01)

Készíts vizualizációt a piros-fekete fához! Látható legyen a csúcsok kulcsa, színe, illetve a szülő-gyerek kapcsolatok a fában!

Nem muszáj grafikus megjelenítést csinálni, jó a konzolos is. Akár el is lehet forgatni a fát. Példa:

33. Feladat (G07F02)

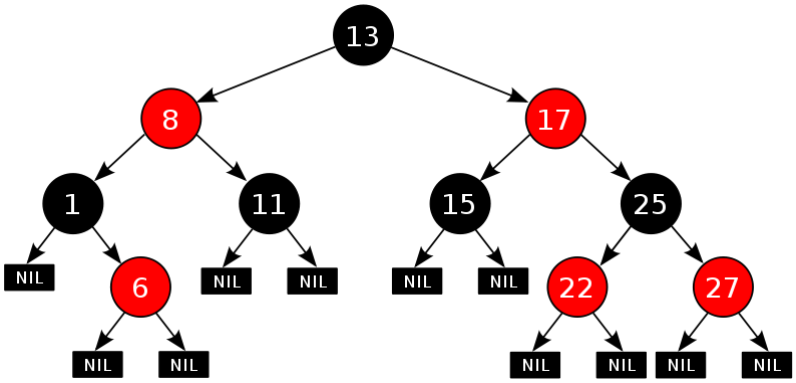
- Készíts egy teszt programot, amely összehasonlítja a Bináris Kereső, az AVL illetve PF fa használatát!
- Ugyanazokat az elemeket (milliós nagyságrendű darabszám) szúrd be a BinKer, az AVL illetve a PF fába is és nézd meg, hogy hogyan változnak a fák magasságai (ezer, tízezer, százezer, millió … elem után).
- A következő teszteket tudja elvégezni a program:
- Elemek növekvő/csökkenő sorrendben történő beszúrása
- Random elemek beszúrása

34. Feladat (G07F03)

- Egy map segítségével készítsetek statisztikát egy nagyon hosszú ékezetek nélküli (ASCII) szöveg szavainak előfordulására.
- **Bemenet**:
  - a program argumentumként egy fájlnevet fogadjon, amely a feldolgozandó szöveget tartalmazza.
  - a szövegben található írásjegyeket ne vegyétek figyelembe, továbbá kis és nagy betű sem számít. A kiadott kódban láthattok egy módszert a szöveg egy parszolási lehetőségére.
- **Kimenet**:
  - **a)** irassátok ki a szavakat ábécé rendben (mellé az előfordulásuk számát)
  - **b)** irassátok ki a szavakat az előfordulásuk számának **csökkenő** sorrendjében, ha több szám ugyanannyiszor szerepelt, akkor ezeket soroltassátok fel a szám mellett (lásd pl.)

- **Segítség**: ha a piros-fekete fa node osztályába egy kulcs-érték párost helyeztek és a kulcs szerint

```
          (27)
      [25]-+
      |    (22)
   (17)-+
    |   [15]
[13]-+
    |   [11]
   ( 8)-+
      |    ( 6)
      [ 1]-+
```



rendeztek, akkor egy map adatstruktúrát kaptok, (**pontosan úgy, mint az előző háziban**)

- Használnotok kell két mapet (két piros-fekete fát).
  - **a) elvárt kimenet pl.:** a ---> 5 against ---> 1 all ---> 1 and ---> 9 appear ---> 2 article ---> 1 as ---> 3 at ---> 1 been ---> 1
  - **b) elvárt kimenet pl.:**12 --> of, the 9 --> and 8 --> by 7 --> d, our, to 5 --> a, this, which 4 --> fortinbras, his, in 3 --> as, so

35. Feladat (G07F04)

A **map** egy rendezett asszociatív konténer, mely kulcs-érték párokat tárol, ahol a kulcsok egyediek.

A feladat egy ilyen **map** osztály implementálása a Piros-fekete fa segítségével.

Legyen lehetőség a **map** osztálynál

- új elemet felvenni(beszúrás)
- elemet törölni
- Adott elemet(kulcs) lekérdezni
- kilistázni a tartalmát

A kulcs és az érték, lehet integer, string, és character is. (Template használata)

Tizedik gyakorlat

36. Feladat (G08F01)

Implementáld az előadáson tanult leszámlálásos rendezést int-ekre!

37. Feladat (G07F02)

n darab egyenletesen eloszló pont a síkon, az egységkörön belül *x* és *y* koordinátájukkal van megadva. Rendezd ezeket az origótól való távolságuk szerint növekvő sorrendbe edényrendezés segítségével!

38. Feladat (G07F03)

Írj egy olyan radix rendezést, amely tetszőleges (akár különböző) hosszúságú szövegeket rendez sorba!

39. Feladat (G07F04)

A felhasználótól kérd be: *n, min, max*

Generálj egy *n* hosszú random sorozatot *min* és *max* közötti elemekkel!

Rendezd az eddig tanult rendezőkkel és hasonlítsd össze a futásidejüket!

Az összehasonlításos rendezőnél számold meg az elemek közötti összehasonlítások számát!

40. Feladat (G07F05)

Egy hangeszközt a következő formátumú bemenettel tudunk vezérelni: egy sorban egy időpont van mikroszekundumban, egy utasítás: KI vagy BE, illetve a hang frekvenciája, szóközzel elválasztva (a számok egészek). Egy ilyen sor a megadott frekvenciájú hangot a megadott időpontban ki- vagy bekapcsolja.

Készíts programot, amely több szólamot össze tud eggyé olvasztani! Kérd el a felhasználótól a szólamok fájljainak nevét, és írd ki az eredményt ugyanebben a formátumban a megadott fájlba!

Készítsd fel a programot, hogy képes legyen 1000 szólamot, egyenként 1000000 paranccsal feldolgozni!

Kiskérdések

Első gyakorlat – Elmélet

01. Add meg a típus modern definícióját!
02. Mire használatos az adattípus absztrakt leírása? (Mire jó és mire nem?)
03. Add meg az adatszerkezet definícióját!
04. Mit takar egy absztrakt adattípus axiomatikus leírása?
05. Mi az adatszerkezet definíciója? Adj legalább 3 szempontot, ami szerint csoportosíthatók!
06. Mit jelent az, hogy egy adatszerkezet dinamikus?
07. Mit értünk absztrakt adatszerkezet alatt? Mutass példát ADS szintű leírásra!
08. Mit jelent az, hogy egy adatszerkezet statikus?
09. Mi a típus specifikáció, mit kell megadni hozzá?
10. Mi a típus, mit kell megadni hozzá?
11. Mit jelent a típus specifikációjának funkcionális megadása?
12. Mit jelent a típus specifikációjának axiomatikus megadása?
13. Milyen szempontok szerint lehet osztályozni az adatszerkezeteket?
14. Mi a különbség a pointer és a referencia között? Milyen értékeket tárolnak?
15. Milyen típusabsztrakciós szinteket ismersz? Mik a fő tulajdonságaik?

Első gyakorlat – C++

01. Mit ír ki az alábbi program a képernyőre?

```
int& fv(int& p1, int* p2, int p3)
{
     p1++;
     *p2 = p1 + p3;
     p3--;
     return p1;
}

int main(int argc, char *argv[])
{
    int a = 5;
    int c = 7;
    int* b = &c;

    b = &(fv(a,b,c));
    (*b)--;

    cout << a << "," << *b << "," << c;

    return 0;
}
```

02. Részletezd a c++ függvények esetén a paraméterátadás mechanizmusát! (Térj ki a lehetséges esetekre!)
03. Az alábbi utasítások közül jelöld, hogy mely(ek) helyes(ek) és mely(ek) nem?

```
int tomb[6];
tomb[1] = 10;
tomb[6] = 28;
*tomb = 11;
*(tomb+2) = 4;
```

```
int * c = tomb;
c+=2;
*c = 4;
tomb[*c] = 1;
```

04. Az előző feladat során használt tomb változó tartalmát írd le a kód lefutása után. (Feltételezzük, hogy csak a helyes utasításokat vesszük figyelembe.)
05. Az alábbi programkód hibát (hibákat) tartalmaz. Keresd meg őket és adj javaslatot a kód módosítására. (Figyelem! A kódban nemcsak szintaktikai hibák vannak, hanem programozás-elméletiek is.)

```
/**
 * ZH Feladat, author: kami.
 */
#include <iostream>
using namespace std;

/**
 * A függvény első és második paraméterét megváltoztatja és az
 * első paraméter négyzetét és a kapott paraméter szorzatát
 * adja vissza a függvény hívásakor.
 */
void negyzetesszorzat(int p1, int p2)
{
     p2 = p1 * p2;
     p1 = p1 * p1;
}

/**
 * Main függvény az Adatszerkezetek és algoritmusok tantárgy
 * 2008/2009 őszi féléves kurzusának első nagy zh-jához.
 */
int main()
{
     int a = 100;
     int *p = a;
     int &b = a;
     cout << a << ' ' << b << ' ' << *p << endl;
     negyzetesszorzat(a, b);
     cout << a << ' ' << b << ' ' << *p << endl;
     return 0;
}
```

06. Mit ír ki az alábbi programkód a képernyőre? A sorok mellé írd fel, hogy a sor futásakor mi abban a sorban található változó aktuális értéke!

```
#include <iostream>

using namespace std;

int* fv12(int& p1, int* p2, int p3)
{
     p1--;
     *p2 = p1 + p3;
```

```
     p3--;
     return &p1;
}

int& fv34(int& p1, int* p2, int p3)
{
     p1++;
     *p2 = p1 - p3;
     p3++;
     return p1;
}

int main(int argc, char *argv[])
{
    int a = 5;
    int c = 7;
    int* b = &c;
    int d = 4;
    int f = 9;
    int* e = &f;

    b = fv12(a,b,c);
    (*b)++;
    e = &(fv34(d,e,f));
    (*e)--;

    cout <<a<<", "<<*b<<", "<<c<<", "<<d<<", "<<*e<<", "<<f;

    return 0;
}
```

07. Mit ír ki az alábbi program a képernyőre? Válaszodat röviden indokold! (6 pont)

```
int fv(int p, int* q, int& r)
{
    p++;
    (*q)++;
    r++;
    return r;
}

int main(int argc, char *argv[])
{
     int i = 5;
    int* j = new int(7);
    int& k = i;
    k++;
    cout << i << " " << *j << " " << k << endl;
    cout << fv(i,j,k) << endl;
    cout << i << " " << *j << " " << k << endl;
```

```
    cout << fv(i,j,*j) << endl;
    cout << i << " " << *j << " " << k << endl;
    return 0;
}
```

Második gyakorlat – Elmélet

01. Mit jelent a láthatóság? Milyen láthatósági szinteket ismersz?
02. Mi a „this” pointer? Miért van rá szükség?
03. Mit jelent az osztályváltozó, osztálymetódus?
04. Mit jelent a példányváltozó, példánymetódus?
05. Mi az a polimorfizmus?
06. Mire használjuk a virtual kulcsszót C++-ban?
07. Mi az a dinamikus összekapcsolás?
08. Mire használjuk a static kulcsszót C++-ban?

Harmadik gyakorlat – Elmélet

01. Adott egy nyolc méretű statikus sor, helytakarékos implementációval. A következő feladatokat hajtsuk végre: in(7), in(4), in(6), in(9), in(1), in(4), out(), out(), out(), in(4), in(5), in(2), in(3), in(9). Vezesd végig ábra segítségével a műveleteket! Minden esetben jelöld a head és tail indexet! Mi történik in(8) esetén? Mi történik out esetén?
02. Mutasd be a lengyelformára alakítás algoritmusát.
03. Mi a fő különbség a statikusan és dinamikusan implementált verem/sor között?
04. Írd le a verem ADT műveleteinek axiomatikus leírását! (Vagyis, hogyan viselkedik a verem, ha az egyes műveleteit végrehajtjuk?)
05. Add meg azt az algoritmust, amely egy matematikai kifejezés esetén eldönti, hogy helyes-e benne a zárójelezés.

Negyedik gyakorlat – Elmélet

01. Mi a szekvenciális adatszerkezet definíciója?
02. Mit jelent a statikus reprezentáció?
03. Mit jelent a dinamikus reprezentáció?
04. Mi a különbség a fejelemes és a fejelemet nem használó láncolt ábrázolás között?
05. Miket tárolunk egy node-ban kétirányú illetve egyirányú lista esetén?
06. Mi a lista műveletei?
07. Milyen műveletek esetén hatékony a kétirányú láncolás?
08. Add meg az egyirányú láncolt lista insertBefore műveletének struktogramját vagy pszeudokódját! A listának nincs más művelete implementálva, így hivatkozni másikra nem lehet. Feltételezheted, hogy a listában legalább három elem van.
09. Add meg az egyirányú láncolt lista removeLast műveletének struktogramját vagy pszeudokódját! A listának nincs más művelete implementálva, így hivatkozni másikra nem lehet. Feltételezheted, hogy a listában legalább három elem van.

Ötödik gyakorlat – Elmélet

01. Add meg egy bináris keresési fában a balra-forgatás algoritmusát!
02. Írd le a globális maximumkeresés algoritmusát bináris keresőfában!
03. Add meg a postorder és az inorder bejárás algoritmusát bináris kereső fa esetén!
04. Határozd meg a törlés eseteket a bináris keresési fában. Írd le a törlés algoritmusát!
05. Egy bináris keresőfában tárolunk prímszámokat. Az egyik fa elem kulcsa a 197. Hogyan találod meg a fában azt a prímszámot, amelyik a nála kisebbek közül a legnagyobb és szerepel a fában? Add meg az algoritmust pszeudo, vagy c++ kóddal. Figyelj, hogy minden esetet kezelj!

06. Egy bináris kereső fából kitörlünk egy csúcsot. Milyen esetek fordulhatnak elő és hogyan őrizzük meg a bináris fa tulajdonságait?
07. Írd le rákövetkező elem meghatározásának algoritmusát bináris keresési fában! (Add meg egyértelműen a balgyerek-szülő-jobbgyerek relációt!)
08. Mit jelent a bináris keresési fa fogalma? (Mindhárom részét definiáld külön!)
09. Írd le egy tetszőleges bináris keresési fában való jobbra forgatás pszeudo vagy c++ kódját, amennyiben az alábbi Node osztály és függvényszignatúra definiált: (Ügyelj a speciális esetekre.) (Az x-et és szülőjét forgatjuk, valamint x és szülője létező elem.)

```
class Node
{
int value;
Node * parent;
Node * leftChild;
Node * rightChild;
}
...
void rotateRight(Node * x);
```

Hatodik gyakorlat – Elmélet

01. Mit jelent az, hogy egy fa kiegyensúlyozott? (*extra: tökéletesen kiegyensúlyozott?*)
02. Melyek az AVL fa tulajdonságai?
03. Milyen összefüggés van az AVL fa csúcsszáma (***n***) és magassága (***h***) között?
04. Rajzolj le egy 4 magasságú, minimális csúcsszámú AVL fát!
05. Rajzolj le egy 4 magasságú, maximális csúcsszámú AVL fát!
06. Mi az AVL fa maximális és mi a minimális magassága 14 node esetén? Válaszodat indokold! (Pontos értékeket kérünk.)
07. Szúrd be a következő elemeket ilyen sorrendben egy AVL fába: ***6, 9, 10, 5, 7, 8***
    Az esetleges forgatások előtti és utáni állapotokat is rajzold le!
08. Szúrd be a következő elemeket ilyen sorrendben egy AVL fába: ***5, 3, 1, 0, 7, 6***
    Az esetleges forgatások előtti és utáni állapotokat is rajzold le!
09. Egy AVL fába beleszúrjuk az alábbi elemeket: ***10, 14, 20, 3, 1, 12, 9, 7*** Rajzold le a lépéseket!
10. Mit jelent a (++,+) szabály? Milyen másik szabályra vezet vissza a helyreállítás során az algoritmusa? Vázold az algoritmust!
11. Mit jelent a (++,-) szabály? Milyen másik szabályra vezet vissza a helyreállítás során az algoritmusa? Vázold az algoritmust!
12. Mit jelent a (--,-) szabály? Milyen másik szabályra vezet vissza a helyreállítás során az algoritmusa? Vázold az algoritmust!
13. Mit jelent a (--,+) szabály? Milyen másik szabályra vezet vissza a helyreállítás során az algoritmusa? Vázold az algoritmust!

Hetedik gyakorlat - Elmélet

01. Írd fel a piros-fekete fa szabályokat!
02. Írd le egy elem beszúrásának a menetét a piros-fekete fába!
03. Miért használunk AVL fák helyett piros-fekete fákat. Mik az előnyök, hátrányok?
04. Írd le egy elem törlésének a menetét a piros-fekete fából!

Tizedik gyakorlat - Elmélet

01. Hasonlítsd össze a Batcher sortot és a merge sortot (hasonlóságok, különbségek, előnyök, hátrányok, használhatóság, könnyű kódolhatóság szempontjából)!
02. Hasonlítsd össze a külső és a fésüléses rendezést (hasonlóságok, különbségek, előnyök, hátrányok, használhatóság, könnyű kódolhatóság szempontjából)!
03. Mi az a lexikografikus sorrend?

04. Mi az az összehasonlításos rendező, mennyi az összehasonlításos rendező komplexitása legjobb esetben?
05. Mi az a konzervatív rendező?
06. Mi az az edényrendező? Mikor alkalmazhatjuk? Mennyiben jobb vagy rosszabb, mint a többi rendező?
07. Mi az a radix rendezés? Mikor alkalmazhatjuk? Mennyiben jobb vagy rosszabb, mint a többi rendező?
08. Meg tudod-e valósítani a radix rendezőt a C++ `std::sort`-ja segítségével? Miért?
09. Fésüld össze a következő (már rendezett) sorozatokat: (1, 3, 5, 9, 17, 19, 20) és (2, 4, 7, 8, 18)!
10. Rendezd merge sort segítségével a következő sorozatot: (8, 7, 2, 3, 5, 6)!
11. Rendezd edényrendezés segítségével a következő sorozatot: (8, 7, 2, 3, 5, 6)!
12. Írd le a merge sort pszeduokódját és struktogramját!
13. Írd le az edényrendezés pszeudokódját és struktogramját!
14. Írd le a Batcher sort PP_Merge eljárásának pszeudokódját és struktogramját!
15. Írd le a radix rendezés pszeudokódját és struktogramját!