```
/*************************************************************************
 *
 * STL és iostream gyorstalpaló
 *
 * A C++ tartalmaz csomó olyan adatszerkezetet és osztályt, amire minden programnak szüksége
 * lehet. Ilyenek a sztring, a duplán láncolt lista, a dinamikus tömb és még sok egyéb.
 * Ezt az osztály gyűjteményt STL-nek, "Standard Template Library"-nek hívják.
 * A programozást nagyban megkönnyítik, mert ezek a C++-ban szabványosak, bármikor
 * használhatóak, nem kell őket újra megírni.
 *
 * Ezeknek az osztályoknak a részletes megértéséhez szükség van eddig nem bemutatott C++
 * nyelvi elemek ismeretére is (pl. osztály sablonok), ezért általában a félév végén
 * szoktak szerepelni a tananyagban. Az egyszerűbb felhasználáshoz azonban, amilyen például
 * a nagy házi feladatokhoz is elegendő, ez nem olyan lényeges még.
 *
 * A házi feladatot már többen elkezdték csinálni. A legtöbb feladat nem valamilyen
 * ilyen alapvető osztály létrehozására koncentrál, és ilyenkor igazából nem éri meg
 * gürcölni a saját, kézzel megvalósított dinamikus sztringgel (char* rémálom), saját
 * dinamikus tömbbel stb. Hogy az energia ne ezeknek a megvalósítására menjen el, röviden
 * leírom, milyen beépített tároló osztályokat ad a C++. Ennek az iromány nak az
 * áttanulmányozása után sokkal könnyebb lesz a házit elkezdeni.
 *
 * Az STL részletesebb dokumentációja elérhető pl. a http://cplusplus.com helyen.
 *
 * Megj.: az osztályok mind az std névtérben vannak; én mindig ki is írom ezt. Nem
 * szeretek using namespace-elni (vannak hátrányai), de ez ízlés kérdése.
 *
 * czirkos@eet.bme.hu
 *
 */

/*************************************************************************
 *
 * std::string, a beépített string típus
 * Dinamikusan átméreteződő, okos string, az összes elképzelhető
 * operátora átdefiniálva.
 *
 */
#include <string>

std::string s;      // Üres stringet hozunk létre. A méretével nem kell foglalkozni.
std::string m("Hello vilag!");
std::string a("alma"), k("korte");

s="Hello";          // Beállítjuk valamilyen értékre.
s+=" vilag!";       // Hozzátoldunk valamit.
std::cout<<s;       // Kiírjuk a konzolra.

if (s==m)           // Összehasonlítjuk; teljesülnie kell a feltételnek.
    std::cout<<"egyformák!";
if (a<k)
    std::cout<<a<<" (kvázi) előbb van az ABC-ben.";

s[0]='h';           // hello vilag!
s.erase(2, 3);      // Töröl a belsejéből: "he vilag!"
s=m.substr(2, 3);   // s új értéke: m belsejéből egy darab, "llo"
int i=a.find("lm"); // Keresés: 1-et ad vissza, mert "alma" szóban ott van az "lm"
a.insert(3, "er");  // Beszúrás: "alma"-ból almera lesz




/*************************************************************************
 *
 * std::vector, átméretezető egydimenziós tömb.
 * Ez egy osztály sablon. A tömb bármilyen elemeket tartalmazhat; a
 * tartalmazott típusokat <> között kell megadni. Például egész számokat
 * tartalmazó tömb: std::vector<int> t;
 *
 */
#include <vector>

std::vector<int> t;         // Tömb, ami egyelőre üres.
std::vector<double> d(50);  // 50 elemű double tömb.
```

```
d[23]=3.14;                 // Egyértelmű
t.push_back(1);             // A push back mindig a tömb végére rak egy új számot,
t.push_back(2);             // és meg is növeli a tömb méretét.
t.push_back(7);
                            // Kiírjuk a tömb tartalmát. Az aktuális méretét a size() adja meg.
for (int i=0; i<t.size(); ++i)
    std::cout<<t[i]<<' ';
std::cout<<std::endl;
t.clear();                  // Kitörli a tömböt, újra üres lesz.

d[60]=10.3;                 // NEM SZABAD!!! Továbbra is 0..49 között indexelődik az 50 elemű tömb.
d.at(120)=0.012;            // Ugyanaz, mint az indexelő, csak futási időben ellenőrzi, hogy nem
                            // hivatkozunk-e nem létező indexre. Ha igen, akkor hibát dob.

#include <algorithm>        // rendezéshez
#include <cstdlib>          // rand() véletlenszámokhoz

t.resize(100);              // Méretezzük át a tömböt 100 eleműre.
srand(time(0));             // Véletlenszám generátor indítása
for (int i=0; i<t.size(); ++i)
    t[i]=rand()%200;        // Minden elem: véletlenszám 0..199 között
std::sort(t.begin(), t.end());      // Rendezzük növekvő sorrendbe az egész tömböt,
                                    // az elejétől a végéig.
for (int i=0; i<t.size(); ++i)      // Írjuk ki az összes elemet.
    std::cout<<t[i]<<' ';
std::cout<<std::endl;




/*************************************************************************
 *
 * std::list, kétszeresen lista osztály, tetszőleges típusú adattagokkal.
 *
 */
#include <list>
#include <cctype>               // toupper-höz

std::list<std::string> szavak;  // stringekből álló lista (!)
szavak.push_back("alma");       // alma  (... került az üres lista végére)
szavak.push_back("korte");      // alma, korte
szavak.push_front("málna");     // málna, alma, korte   (elejére teszi!)
szavak.push_front("villa");     // villa, málna, alma, korte
std::cout<<szavak.size()<< " szót tartalmaz a lista.\n";
szavak.remove("villa");         // kitöröljük a kakukktojást: málna, alma, korte
std::cout<<"Az első elem: "<<szavak.front()<<std::endl;
std::cout<<"Az utolsó elem: "<<szavak.back()<<std::endl;

std::list<std::string> masolat;
masolat=szavak;                     // Lemásoljuk a listát. Egy szimpla értékadás megcsinálja!
while (!masolat.empty()) {          // Amíg nem üres a lista:
    std::cout<<masolat.back()<<' ';     // Kiírjuk az utolsó elemet...
    masolat.pop_back();                 // és kitöröljük. A lista ezáltal visszafelé íródik ki.
}

// Az egyes tároló osztályokhoz létezik ún. iterátor osztály, amely legegyszerűbb esetben
// arra jó, hogy az összes adaton egy for () ciklussal végig tudjunk menni. Az iterátoron
// értelmezett két legalapvetőbb művelet a ++ (következő elem) és a * (jelenlegi elem értéke).
// Ebből a szempontból ugyanúgy működnek, mint a pointerek, csak sokkal okosabbak azoknál!
// Érdemes megfigyelni, hogy a lista belső felépítéséről semmit nem tudunk, ez el van rejtve
// előlünk. Az iterátor viszont együtt dolgozik a listával; tudja, hogy a lista belsejében
// mi hogyan van megoldva. De azt nekünk nem kell tudni. Legyen az az iterátor dolga.

std::list<std::string>::iterator it;    // Sztringekből álló lista iterátora.
// Indulunk a lista elejéről (begin). Addig megyünk, amíg el nem érjük a lista végét (end).
// Amikor egy adott elemmel elvégeztük a dolgunkat, ugrunk a következőre (++). Ismerős?
for (it=szavak.begin(); it!=szavak.end(); ++it) {
    (*it)[0]=toupper((*it)[0]);         // nagybetűsítjük az első karakterét a szónak
    std::cout<<*it<<' ';                // kiírjuk a szót.
}
```

```
/*****************************************************************************
 *
 * std::set, halmaz osztály, bármilyen típusokat képes tárolni. Adott értékről
 * meg tudja mondani, szerepel-e benne.
 *
 */
#include <set>

std::set<int> s;            // Egész számok halmaza

std::cout<<"Írj be számokat, 0=vége!"<<std::endl;
int i;
std::cin>>i;
while (i!=0) {
    if (s.find(i)!=s.end()) // s.end() jelzi, ha nem találta meg a keresett elemet
        std::cout<<"Ez már benne volt!\n";
    else                    // Berakjuk a halmazba a kapott számot. Természetesen, ha már benne
        s.insert(i);        // volt, amúgy se történne semmi.
    std::cout<<"Most "<<s.size()<<" szám van a halmazban.\n";

    std::cout<<"Kérem a következőt!\n";
    std::cin>>i;
}

// Az iterátorok itt is használhatóak. (És egyébként az összes tárolónál használhatóak
// és ugyanígy működnek! Vektornál, listánál, halmaznál, mindenhol. Nem kell foglalkoznunk
// azzal, hogyan néznek ki belülről!) Az egész számokat tartalmazó halmaz iterátorát a ciklus
// fejlécében deklarálom, a for (int i=0...) mintájára természetesen ezt is lehet.
std::cout<<"A halmaz elemei: ";
for (std::set<int>::iterator it=s.begin(); it!=s.end(); ++it)
    std::cout<<*it<<' ';




/*****************************************************************************
 *
 * std::map, asszociatív tömb. Értékeket képez le. Mintha egy tömb lenne,
 * amit nem számmal kell indexelni.
 *
 */
#include <map>

// Sztringeket képezünk le egész számokra; vagyis minden sztringhez rendelünk
// egy egész számot.
std::map<std::string, int> m;

m["Ernőke"]=5;          // Mintha a tömböt indexelnénk, csak épp sztringgel.
m["Pistike"]=7;
m["Orsika"]=4;

std::cout<<"Ernőke, még csak "<<m["Ernőke"]<<" éves vagy."<<std::endl;

// Vigyázni kell, hogy ha olyan névvel indexeljük a map-et, ami még nem szerepel
// benne, akkor létrejön az az elem! És kap kezdeti értéket is, ami intek esetén nulla.
//      std::cout<<m["NincsIlyen"]<<std::endl;
// A fenti sor nullát írna ki, és létrehozna egy NincsIlyen nevű embert!!!
// Ha ellenőrizni szeretnénk, hogy van-e benne, akkor a find függvény használható.
std::cout<<"Kinek szeretnéd tudni a korát? ";
std::string s;
std::cin>>s;
if (m.find(s)!=m.end())
    std::cout<<s<<' '<<m[s]<<" éves."<<std::endl;
else
    std::cout<<"Nincs ilyen név, hogy "<<s<<"!"<<std::endl;

// Persze itt is lehet iterátorunk, amely iterátor viszont nem csak az értéket,
// hanem a kulcsot is visszaadja. it->first a kulcs, it->second pedig az érték.
// Sztringeket egészekre leképező map iterátora:
std::map<std::string, int>::iterator it;
for (it=m.begin(); it!=m.end(); ++it)
    std::cout<<it->first<<" még csak "<<it->second<<" éves."<<std::endl;
```

```
/*************************************************************************
 *
 * iostream, fstream, ... A C++ fájl műveleteit megvalósító osztályai.
 * Jól kombinálhatók az std::stringgel. Részletesebben lásd Izsó Tamás
 * írását: http://www.hit.bme.hu/~izso/stream.pdf
 *
 */
#include <fstream>
#include <iomanip>      // manipulátorok, lásd lent

std::ifstream is;       // input file stream, vagyis olvasunk egy fájlból

is.open("fajl.txt");    // megnyitjuk a fájlt
if (!is) {
    std::cerr<<"Nincs ilyen fájl!";
    // ... meg csinálunk is valami értelmeset ilyenkor
}

// Beolvasunk egy teljes sort. A string szükség szerint átméreteződik!
std::string s;
int i=0;
while (getline(is, s))
    // Számozva kiírjuk a sorokat. std::setw manipulátor: az utána
    // következő kiírt dolgot 5 karakter szélességűre veszi.
    // Mintha printf %5d lenne.
    std::cout<<std::setw(5)<<++i<<". sor: "<<s<<std::endl;
is.close();




/*************************************************************************
 *
 * Komplexebb példa: számoljuk meg, egy szövegfájlban melyik szó hányszor
 * szerepel.
 *
 */
#include <iostream>
#include <map>
#include <string>
#include <fstream>

int main()
{
    std::ifstream is("fajl.txt");   // konstruktor egyből meg is nyitja
    if (!is) {
        std::cerr<<"Nem lehet megnyitni!"<<std::endl;
        return 1;
    }

    std::string s;
    std::map<std::string, int> m;   // sztringeket képezünk le egész számokra
    // Beolvasunk egy szót...
    while (is>>s)
        // és ha sikerült, a map megfelelő számát növeljük eggyel.
        // Ha még nem volt olyan, létrejön 0 értékkel (magyarázat: lásd fent).
        m[s]++;
    is.close(); // Kész.

    std::map<std::string, int>::iterator it;
    for (it=m.begin(); it!=m.end(); ++it) {
        std::cout<<it->second<<"x szerepelt ez: ";
        std::cout<<it->first<<std::endl;
    }
}
```